CONTRIBUTIONS TOWARDS IMPROVING THE USEFULNESS OF PARTIAL VISUALIZATIONS IN PROGRESSIVE VISUAL ANALYTICS

MARIUS HOGRÄFER

PHD THESIS

October 2022

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CONTRIBUTIONS TOWARDS IMPROVING THE USEFULNESS OF PARTIAL VISUALIZATIONS IN PROGRESSIVE VISUAL ANALYTICS

MARIUS HOGRÄFER

A Thesis in Partial Fulfillment of the Requirements for the Ph.D. Degree
Presented to:

Department of Computer Science
Faculty of Natural Sciences
Aarhus University

October 2022

Marius Hográfer: *Contributions Towards Improving the Usefulness of Partial Visualizations in Progressive Visual Analytics* PhD Thesis, © October 2022

The road to wisdom? Well, it's plain
And simple to express:
Err
and err
and err again,
but less
and less
and less.

— Piet Hein

ABSTRACT

Progressive Visual Analytics (PVA) is a recent approach for gaining insights into massive datasets, providing analysts with immediate results of computations that would otherwise run for much longer. This is done by showing early, partial visualizations of these computations that refine over time, rather than waiting until they complete before showing analysts any results. The related work has shown analysts in PVA to clearly outperform those using non-progressive systems in terms of completion time, while finding comparatively many insights. A defining challenge for the success of PVA is that these partial visualizations are as useful as possible, in that they "show what analysts need to see", which is an inherently subjective assessment that must be made by the analysts in their particular analysis context. Consequently, only analysts can improve this usefulness by configuring the process to their needs. However, this requires that this process can, in fact, be configured. This desired configurability stands in contrast to the dedicated (i.e., preconfigured) solutions presented in the PVA literature, as they often address a particular analysis scenario and, thus, cannot be further configured.

In response, this thesis contributes configurable approaches that enable analysts to bring in their subjective expertise to improve the usefulness of partial visualizations in PVA in terms of their expressiveness, effectiveness, and efficiency. This is achieved by extending analysts' control over the data shown in partial visualizations. First, this thesis contributes tailorable sampling for PVA, which allows analysts to ensure that partial visualizations express the desired information as early as possible, in a way that can be adapted to new insights without restarting the computation. Then, it contributes configurable strategies for context-dependent Degree of Interest (DOI) functions enabling their effective use in PVA, to allow extracting the desired information as early as possible in partial visualization. Third, the thesis contributes a steering approach that poses minimal requirements on the analysis scenario, allowing analysts to focus their computations on relevant data subspaces.

Overall, with these contributions in place, this thesis provides a step towards a fully configurable PVA process that can be tailored by analysts based on their particular needs, in order to make partial visualizations as useful as possible as early as possible.

ABSTRACT (IN DANISH)

Progressive Visual Analytics (PVA) er en nyudviklet metode til at opnå indsigt i meget store datasæt, som giver analytikere øjeblikkelige resultater af beregninger, der ellers ville skulle køre i meget længere tid. Dette gøres ved at vise tidlige delvise visualiseringer af disse beregninger, som så bliver forfinet over tid, frem for at vente indtil beregningerne er færdige, før resultaterne bliver tilgængelige for analytikeren. Tidligere arbejde har vist, at analytikere, der bruger PVA, på væsentligt kortere tid gør lige så mange opdagelser i deres data som analytikere, der bruger ikke-progressive systemer. En fundamental udfordring for PVA's succes er at gøre de delvise visualiseringer så nyttige som muligt — altså sørge for at de "viser det, som analytikeren har brug for at se", hvilket er en subjektiv vurdering, der må foretages af analytikeren i vedkommendes specifikke analysekontekst. Dermed er det kun analytikeren, der kan forbedre PVA's brugbarhed ved at konfigurere processen til sine behov. Men det kræver, at processen rent faktisk kan konfigureres. Behovet for at konfigurere PVA-processen står i kontrast til de prækonfigurerede løsninger, der typisk præsenteres i PVA-litteraturen, da disse ofte adresserer et bestemt analyse-scenarie og derfor ikke kan konfigureres yderligere. Denne afhandling adresserer dette ved at bidrage med konfigurerbare løsninger, der lader analytikeren gøre brug af sin subjektive ekspertise og forbedre de delvise visualiseringers udtrykskraft og effektivitet og dermed forbedre PVA's anvendelighed.

Dette opnås ved at udvide analytikerens kontrol over den data, som vises i de delvise visualiseringer. Til at starte med præsenterer afhandlingen PVA-sampling, der kan skræddersys, så analytikeren kan sikre, at de delvise visualiseringer udtrykker den ønskede information så tidligt som muligt, på en måde der kan tilpasses til nye opdagelser, uden at beregningen skal startes på ny. Dernæst præsenterer afhandlingen konfigurerbare strategier til kontekstafhængige interesse grads funktioner (Degree of Interest functions/DOI functions) og muliggør derved effektiv anvendelse af dem i PVA, for at udtrække den ønskede information så tidligt som muligt i delvise visualiseringer.

Endeligt præsenterer afhandlingen en styremetode, der stiller minimale krav til analysesituationen og lader analytikeren fokusere sine beregninger på relevante data-underrum. Som helhed er denne afhandling et skridt hen imod en fuldt konfigurerbar PVA-proces, der kan skræddersys af analytikere baseret på deres specifikke behov, således at delvise visualiseringer kan gøres så nyttige som muligt så tidligt som muligt.

PUBLICATIONS

The contributions of this thesis are based on the following publications and manuscripts:

PUBLISHED PAPERS

P1 **Marius Hogräfer**, Jakob Burkhardt, and Hans-Jörg Schulz. *A Pipeline for Tailored Sampling for Progressive Visual Analytics*. Proceedings of the International EuroVis Workshop on Visual Analytics (EuroVA), 2022, pp.49-53 doi:10.2312/eurova.20221079.

P2 **Marius Hogräfer**, Marco Angelini, Giuseppe Santucci, and Hans-Jörg Schulz. *Steering-by-Example for Progressive Visual Analytics*. ACM Transactions on Intelligent Systems and Technology (TIST), Volume 13, Issue 6, 2022, pp.96:1–96:26 doi:10.1145/3531229.

MANUSCRIPTS

M1 **Marius Hogräfer** and Hans-Jörg Schulz. *Tailorable Sampling for Progressive Visual Analytics*. In submission at IEEE Transactions on Visualization and Computer Graphics.

M2 **Marius Hogräfer**, Dominik Moritz, Adam Perer, and Hans-Jörg Schulz. *Strategies for Enabling Degree-of-Interest Functions for Progressive Visualization*. In preparation for submission to EuroVis 2023.

ACKNOWLEDGMENTS

This PhD thesis is the outcome of three exciting years at Aarhus University, and I owe thanks to all the many amazing people that were part of that journey.

First and foremost, I want to thank my advisor, Hans-Jörg, for inviting me to Denmark and for introducing me to Progressive Visual Analytics. I am infinitely grateful for your guidance and support over the last three years. I have grown a lot as a young researcher, and this would not have been possible without you occasionally nudging me back onto the right track. This PhD has been quite an experience, and I am thankful for all our work together.

I also thank all my co-authors for our fruitful collaborations.

Many thanks are also due to my colleagues here at AU, in particular my fellow UBI and CoCHI people, who warmly welcomed me when I arrived here and provided me with an open and accepting atmosphere for conducting my research, including the countless lunches and Friday bars we shared together. I also want to thank Adam, my host during my virtual stay-abroad, as well as everybody in the DIG lab that welcomed me (virtually) at CMU.

Thanks are due to my friends in Aarhus, who supported me along the way, including (but not limited to) Wenkai and Jana, the world's best office mates; Nathalie for your tremendous help and support during thesis writing (and the kardamommesnurrer); Philip, for your inspirational pep-talks; Marcel, for our mutual venting about PhD life and Deutsche Bahn journeys; Henriette for the cumulative marathons we ran along the coast of Aarhus that helped me clear my head from time to time; Noga for our coffee breaks; Ida, Matthias, Lars, Peter, and Penny the one and only Ceres gang. Thank you, Tommaso and Virginia, i miei italiani, for enduring late-night typing noises during these last few months, and mille grazie for all the comfort foot (especially the pizza), TV series, and conversations we shared together.

And thank you, Sofie, for always being there for me, and for supporting me (and putting up with me) during these last few months.

Lastly I also want to thank my family, who always supported me from afar. Mama, Papa, Tobi, Heike, Kalle; danke für eure undendliche und bedingungslose Unterstützung, bei all den merkwürdigen Dingen, für die man mich bezahlt. Oma und Opa, danke für die Pakete voll mit dringend benötigter Nervennahrung, und dafür, dass ihr mich immer auf dem Laufenden gehalten habt. I could not have gone through with this PhD journey without you.

Marius Hogräfer,
Aarhus, October 2022.

CONTENTS

ACRONYMS

PVA	Progressive Visual Analytics
VA	Visual Analytics
DOI	Degree of Interest
InfoVis	Information Visualization
MDS	Multi-dimensional Scaling
PCA	Principal Component Analysis
SAX	Symbolic Aggregate approXimation
t-sne	t stochastic neighbors encoding
UMAP	Uniform Manifold Approximation

Part I

OVERVIEW

1 INTRODUCTION

1.1 MOTIVATION

Visual Analytics (VA) is a research field exploring the interactive visual analysis of data [64]. The general idea behind VA is to leverage the cognitive skills of human analysts on one side and the processing power of modern computers on the other to facilitate sense-making on large complex datasets. The interaction between them runs as a dialogue: The computer performs the analytic processing and presents results through an interactive visualization, which human analysts can then efficiently interpret and reason about, bringing in common sense and domain knowledge. Based on the insights they gather, analysts can then interact with the visualization to reconfigure the computation, prompting the computer to rerun the analysis and update the visualization. This continues until analysts eventually complete their task. In order for VA to be effective, though, the dialogue with the computer must be fluid, in that the visualization should update quickly after analysts perform their interaction, as analysts otherwise lose their "flow" [33] or get distracted. An obvious challenge here is that running complex computations on large datasets even with modern hardware can take a long time (minutes, hours, even weeks), which inherently reduces the successfulness of VA.

A recent approach to this problem is Progressive Visual Analytics (PVA) [107]. Rather than waiting until these long-running computations have completed before showing any results to analysts, in PVA, partial visualizations show *early* results of the computation, bringing down waiting times to interactive rates. By tightening the feedback loop, PVA allows analysts to maintain their flow and to draw insights based on early results, even when the computation is long-running.

The reason why this approach is successful is that patterns found in the final result can in many cases already be observed or approximated early on in the computation. In the words of Angelini et al., PVA in principle allows analysts to interact with the partial visualization, *"as if interacting with the final result"* [5]. An example of four partial visualizations is depicted in Figure 1. The patterns of the result after the computation has completed can already be observed at 50% and partially even at 25%. In addition to early insights as in the example, PVA has other benefits for the analysts. For example, analysts can identify errors in the computation early from partial results and stop the computation without wasting time, and they can gain insights into the inner workings of otherwise black-box computations.

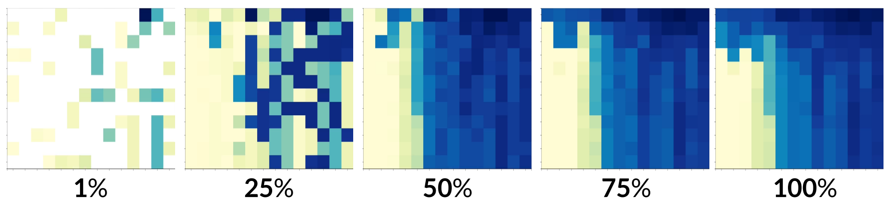


Figure 1: An example of a progression that incrementally processes partitions of the data, showing how even at early stages of the progression, patterns from the final result become apparent. Adapted from Loeschcke et al. [74].

PVA can, however, also be unsuccessful. This is because in order for PVA to work, the partial visualization must be *useful* for analysts. In turn, whenever that is not the case, for example, as no clear patterns form or when they never stabilize until the computation completes, the progressive approach is not necessarily beneficial. Or worse: PVA can also mislead analysts into drawing false conclusions from patterns that are only present in the partial results, but not in the final visualization. This is, of course, detrimental to the analysis.

A defining challenge for PVA is, thus, to ensure that partial visualizations become as useful as possible as early as possible.

The contributions of this thesis focus on this general challenge. Current PVA literature mostly views usefulness in terms of generating a visualization that is as similar to the final visualization as possible [41]. In practice, however, usefulness is often a more nuanced trait, in that some parts of the partial visualization are more relevant than other parts to extract the information needed for completing a particular analysis task. This is evident in the conceptual notion that in PVA, a partial visualization is *"only useful if it shows what the analyst needs to see"* [6], which clearly highlights the need for subjective interpretation of usefulness.

At the same time, this subjective and scenario-dependent notion of usefulness implies that the process of generating partial visualizations needs to be configurable to that scenario. This is because only analysts themselves know, what makes the partial visualization useful in their particular analysis scenario. In addition to that, analysts' interests can dynamically change at runtime as they learn more and more about their data, and consequently the usefulness of the partial visualization needs to remain configurable at runtime. This means that configuring a useful partial visualization is not just a one time operation made by a system designer, but it is an ongoing process *during* the analysis by the analysts actually using that system. However, many PVA approaches in the field are dedicated solutions to specific analysis problems that can be difficult or impossible to configure, both before and during the analysis. This rigidity poses an obvious challenge whenever "what analysts need to see" — and thereby what makes the visualization useful — changes or does not match with what a system was designed for. In other words, there still exists a clear gap between how usefulness of partial visualizations in PVA is viewed conceptually and how it is achieved in practice.

This thesis addresses this gap by contributing approaches for improving the usefulness of partial visualizations in PVA by configuring the process to analysts' interests. In particular, this thesis makes contributions to the following three criteria:

- Expressiveness (data perspective): express the desired information as early as possible.
- Effectiveness (task perspective): allow extracting the needed information as early as possible.
- Efficiency (resource perspective): the gains from using the visualization outweigh its costs as early as possible.

In doing so, this thesis builds on and expands upon existing notions of what makes partial visualization useful for PVA.

1.2 MAIN CHALLENGES

As outlined above, the usefulness of partial visualizations in PVA needs to be subjectively assessed and configured by analysts based on their particular interests. Usefulness of visualizations in VA can be assessed in terms of how well they cater to a given visualization problem at hand: the data (expressiveness), the task (effectiveness), and the available resources (efficiency) [108, p. 17-18]. This thesis addresses the following research challenges that currently limit the usefulness of partial visualizations in PVA in terms of these criteria, contributing approaches for configuring the PVA process to analysts interests.

1.2.1 *Expressiveness: The sampling cannot be tailored dynamically*

A visualization is characterized as expressive of the underlying data, if it *"express[es] the desired information contained in the data, and only this information"* [108, p.17]. A **problem** for the expressiveness of partial visualizations in PVA is that this "desired information" may not even be present yet in the partial data, which means it simply cannot be expressed. A reason for this to happen is that the sampling responsible for defining the order in which the data gets processed spreads it randomly throughout the progression. A solution to this challenge, therefore, is tailoring the sampling to the user interest, such that it brings out the desired information as early as possible. The **current**

approach is to configure the sampling once before starting the progression, meaning that analysts can no longer configure it later on when their interest changes. However, the user interest does in fact often change during the analysis in PVA with analysts learning more about their data [78], and so this approach clearly limits the expressiveness of partial visualizations. The **research challenge** here is that existing PVA sampling approaches are usually designed to address specific analysis scenarios and, therefore, tailoring them to other scenarios can in itself be difficult, costly, or even impossible — let alone during the analysis. This means that an entirely new approach is needed for sampling in PVA, which can account for this.

This thesis **contributes** tailorable sampling for PVA, which allows dynamically configuring the sampling to the current user interest, thereby increasing the expressiveness of partial visualizations.

1.2.2 *Effectiveness: DOI functions can be infeasible in PVA*

Tominski and Schumann characterize a visual encoding as effective, if it is *"designed such that it allows extracting the information needed for the task"* [108, p.17]. A **problem** for the effectiveness of partial visualizations in PVA is that this information can be quickly buried under the masses of the data that progressive visualization is often used for. For non-progressive visualization, this is commonly overcome by configuring the encoding, such that the most interesting data stands out, while reducing or even completely hiding less interesting data. This is enabled by Degree of Interest (DOI) functions, which compute the "interestingness" of each data item in context of all other data. In PVA this data context changes all the time with each processed sample. This can mean that the same item gets a high interest score at early stages, but as more data gets processed, may lose its importance, or — vice-versa — an "uninteresting" item may gain in importance as it becomes part of a pattern that only forms in later steps of the progression. In response, the **current approach** for utilizing DOI values in PVA is to continuously recompute the interest values for all data with every new sample, which is too time-consuming for many DOI functions. The **research challenge** here is to allow analysts to specify a suitable subset of the data, which on one hand includes as many data items as possible to maintain valid interest values, but on the other only as few as necessary to reduce the computation time. This strongly scenario-dependent specification requires that the solution is flexible enough to capture a variety of subjective requirements.

This thesis **contributes** strategies that allow analysts to tailor the input selection for DOI functions in PVA to increase the effectiveness of partial visualizations.

1.2.3 *Efficiency: Steering mechanisms are restricted to certain scenarios*

Lastly, a visualization is efficient, if *"[t]he gains from using [it] outweigh the computational resources and human effort needed to carry out the analysis"* [108, p.18]. While PVA in general reduces the time before analysts can analyze their data, a remaining **problem** for the efficiency is that time and computational resources may still be wasted processing data that is not relevant to analysts. For example, analysts may identify an emerging pattern in the data that they want to investigate further, but they need to wait until more of *that* data gets processed. The **current approach** to this is to allow analysts to steer the computation, such that it prioritizes unprocessed data that is similar to those in the emerging pattern. The challenge here is that existing steering mechanisms require either a 1-to-1 mapping between data and view space, which not all visualization techniques employ, or they require maintaining a dedicated index structure, which can quickly become too demanding for large datasets. The **research challenge** here lies in designing a novel steering algorithm that overcomes the limitations of existing approaches, while posing minimal requirements on the analysis scenario, allowing analysts to efficiently focus resources on their interests.

This thesis **contributes** a steering mechanism for PVA which only requires a set of relevant items as input to steer the progression towards a data subspace to increase the efficiency of partial visualizations.

1.3 THESIS STATEMENT

This thesis explores the hypothesis that configurable approaches that improve the expressiveness, effectiveness, and efficiency of partial visualizations in PVA enable analysts to bring in their subjective expertise in making them as useful as possible as early as possible. This is based on the idea that usefulness of partial visualizations in PVA should be assessed subjectively by analysts and, therefore, can be improved by providing analysts control over the process to configure it to their interests. Identifying what is desired allows configuring PVA to improve the usefulness of partial visualizations. Consequently, providing configurable approaches is a step towards user-centric design approaches for future PVA systems, which focus on usefulness as a central design goal.

1.4 RESEARCH QUESTIONS AND CONTRIBUTIONS

To support the thesis statement above, this thesis addresses the following research questions.

RQ1: *How can the sampling in PVA be tailored to bring out the desired information in partial visualizations?*

This thesis addresses RQ1 in multiple ways. First, it contributes a characterization of PVA-sampling, distinguishing these approaches from regular sampling in VA, in that sampling in PVA is an ongoing process running in parallel to the analysis, whereas regular sampling is a step performed once before the analysis. It also contributes a pipeline that modularizes PVA sampling into three steps, allowing to tailor the sampling to analysts' interests and in such a way that does not require restarting the analysis. The tailorability of the pipeline is evaluated in a series of examples, demonstrating how each step affects the resulting sample, but also demonstrating how existing samplings can be recreated, recomposed, and tailored to user interests. To demonstrate the user experience, the publicly available tool ProSample is introduced, which implements the pipeline concept and facilitates the interactive comparison of samplings.

RQ2: *How can DOI functions in PVA be continuously updated to allow extracting the information needed for the task from partial visualizations?*

This thesis addresses RQ2 by first contributing a characterization of the challenges limiting the use of DOI functions in progressive visualizations, which are caused by the continuous updates required for producing valid interest scores over changing input data. Based on this, this thesis also contributes a series of strategies for limiting the computational overhead caused by these updates by only selecting specific data items. These strategies allow analysts to tailor what data is included in the DOI computations based on their task, thereby enabling the use of new DOI functions in PVA. These strategies are evaluated in a series of benchmarks, which suggest the use of interest-specific strategies to improve the accuracy of interest values. The interactive visual analysis tool ProInterest is introduced, which adapts these strategies.

RQ3: *How can steering be made more widely applicable across PVA scenarios to reduce the mental and computational efforts of using partial visualizations?*

This thesis contributes a steering approach called steering-by-example that allows prioritizing subspaces of interest in the analysis. In contrast to prior approaches, it only requires a small set of relevant items from that subspace as input. This makes steering-by-example applicable for many visualization techniques beyond 1-to-1 mappings, and it does not require continuous re-binning of the entire unprocessed data space. The approach is evaluated in benchmarks, comparing it to state-of-the-art steering mechanisms, showing a clear improvement in terms of precision in retrieving data from a selected subspace in the next partition. To demonstrate the user experience of steering-by-example, the publicly available tool ProSteer is introduced, which allows to interactively assess the utility of steering-by-example over state-of-the-art steering.

1.5 RESEARCH APPROACH

According to Brooks, conducting computer science resembles the work of a toolsmith, which he describes as symbiotic work of applying scientific and engineering methods [16]. In order to address research questions in the field, Brooks characterizes that work as inherently cyclic: One generally has to build a tool that can be studied, first (which Brooks calls the science perspective), yet in order to be able to build these tools, one also needs to consider the theory (which Brooks calls the engineering perspective). Since this thesis explores *practical* solutions for challenges in a *conceptual* process, its research approach is inspired by this cyclic workflow to address the research questions.

Most contributions came about by considering a solution to a conceptual challenge for useful partial visualizations in PVA, and while constructing tools to demonstrate them, details of these solutions were then fleshed out. Specifically, the contributions of this thesis were facilitated by *ProSample*, a visual comparison tool for pipeline configurations in PVA (see Section 5.4.1), *ProInterest*, a visual interface for configuring progressive DOI functions (see Section 7.7.3), and *ProSteer*, a visual comparison tool for the steering-by-example approach (see Section 8.5). The source code to the tools designed for addressing the research questions is publicly available under open source licenses, to ensure the reproducibility of the reported evaluation results, as well as reusability of the proposed methods.

The contributions towards the thesis statement, thus, came out of a cyclic process similar to the one discussed by Brooks. This is because the process of tool building itself was necessary to evaluate, whether a solution was feasible while revealing detailed challenges, and it also informed further conceptual considerations for solutions. In the case of RQ2 for example, the original goal was to implement a specific DOI function for PVA, which revealed the inherent conceptual challenge that the data context changes progressively. This insight then led to more theoretical work again, exploring general approaches for context-dependent computations, which, in turn, were then implemented in ProInterest. The insights from these implementations then informed the strategies that are also presented in the paper, which were then quantitatively evaluated in the presented benchmarks.

1.6 AUTHORSHIP STATEMENT

While I am the principal author of this thesis, much of my work was done in collaboration with my advisor Hans-Jörg Schulz, and the papers were written in collaboration with the respective co-authors. Below I outline how I contributed to each paper included in this thesis:

TAILORABLE SAMPLING I made significant contributions to the concept and design phases of the initial tailorable pipeline research, and I contributed a majority of the implementation of ProSample. I am the first author of the published paper, which is included in Chapter 5. I also led the extension of this work, leading the concept and design phases, and I was also responsible for extended implementations and benchmarks. I am the first author of the paper, which is included in Chapter 6 of this thesis.

STEERING-BY-EXAMPLE I made significant contributions to the concept and design phases of the project, I contributed a majority of the implementation of ProSteer, and I planned, conducted, and analyzed the expert interviews in Aarhus. I am shared first author of the paper, which is included in Chapter 8 of this thesis.

DEGREE OF INTEREST FOR PVA I led the concept and design phases of the project and I provided the implementation and conducted the benchmarks. I am the first author of the paper, which is included in Chapter 7 of this thesis.

1.7 THESIS OUTLINE

The remainder of this thesis is split into two parts. Part I goes on to provide a background on the related scientific literature in Chapter 2. Based on this, the research challenges for useful partial visualizations are identified and the contributions towards addressing them are summarized in Chapter 3. Chapter 4 concludes the first part by reflecting on the research question in light of the contributions of this thesis and by proposing future research directions.

Part II consists of the papers containing the thesis contributions.

2 BACKGROUND

This chapter discusses the relevant scientific literature, building the theoretical foundation on which the considerations in the following chapters rest. In the first part provides the scientific background on Visual Analytics, leading up to one of its challenges: long computation times. This is then where Progressive Visual Analytics provides a solution and the background on this topic, therefore, forms the second part of this chapter.

2.1 VISUAL ANALYTICS

The contributions of this thesis are situated in the research field of Visual Analytics (VA). Keim et al. define VA as follows: *"Visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision-making on the basis of very large and complex data sets."* [64]. The goal in VA is to get the best out of two worlds: The cognitive skills of the human analyst, who has domain-specific background knowledge about the dataset and can analyze things in context, and the computation power of modern hardware that excel at storing and processing large amounts of efficiently with complex analytic algorithms. In VA a data visualization serves as the two-way interface between user and computation: The visualization presents the analysis results to analysts, and it also allows analysts to interact with the computation to adjust it, which in turn produces a new visualization. This back and forth between computer and analyst distinguishes VA approaches from "regular" (non-interactive) data analysis, and it has been shown to be success-

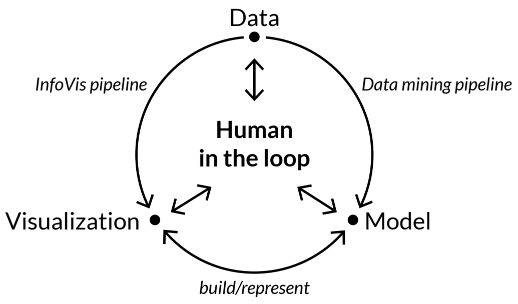


Figure 2: A basic model of the VA process (inspired by the model by Keim et al. [64]). VA puts the "human in the loop" during the analysis, giving analysts full control over the model and data through interactive visualizations.

ful in many domains that require processing large amounts of data like healthcare, fraud detection, and disaster management.

In this section, the related scientific literature on VA is discussed. This both serves as theoretic background for the following chapters and helps to motivate the need for progressive approaches and the research challenges this thesis addresses. In particular, the following aspects are highlighted: the need for fluid interactions, Degree of Interest functions, and the role of regular (non-progressive) sampling in VA.

2.1.1 *Interactive Information Visualization*

Depending on the context, there are different ways to interpret the term "visualization" [22]. This thesis uses it interchangeably to interactive information visualization, which Card et al. define as *"the use of computer-supported, interactive, visual representations of abstract data to amplify cognition"* [17]. With visualization as the interface between computation and analyst in VA, designing an efficient visualization for a particular analysis task becomes an essential Human-Computer Interaction (HCI) challenge underpinning all VA applications. The reason why visualization is successful is that vision as an information channel has the highest bandwidth compared to all other senses, which enables us to immediately recognize certain visual patterns and then interpret them in context of the data [116, pp. 152]. This allows humans to immediately spot a red dot in a sea of blue dots, or distinguish clusters of similar items that are placed closely together, but separately from other clusters. It also allows discovering patterns that are otherwise obfuscated behind descriptive statistics [7], as depicted in Figure 3. The most efficient visual encodings have repeatedly been evaluated experimentally [24, 77], e.g., mapping numerical values to the height of a rectangle or categorical values to categorical colors scales.

Beyond mapping data to graphical symbols, modern computing hardware also makes it possible to also update visualizations based on user input. Analysts can further explore their data, for example, by

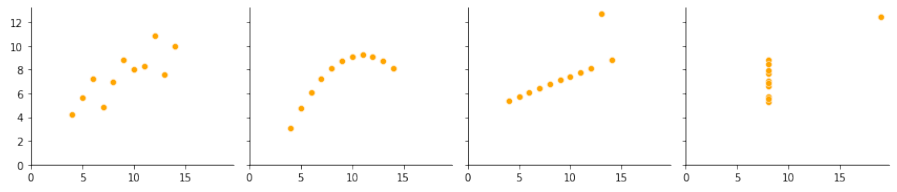


Figure 3: The four depicted datasets have almost identical basic descriptive statistics, meaning that they appear similar when only relying on these metrics. However, a simple visualization reveals their differences immediately [7].

panning and zooming into a scatterplot to enlarge a region of interest, or by highlighting data selected in one plot in another. Through interaction, analysts can take an active role in the visual analysis, which in particular benefits the exploration of large, complex datasets. These interactions are often performed directly on the data using direct manipulation [105]. For example, instead of setting view parameters by manually selecting the range of x and y axes, analysts can use direct manipulation by dragging the visualization with the mouse cursor, or using the scroll wheel to zoom into the region the cursor points to. Direct manipulation reduces the separation between user and data, and thus makes the analysis more direct, accessible, and engaging.

2.1.2 *Ensuring Fluid Interaction in VA*

Designing VA systems is not a trivial endeavor, as the interaction between analyst and system must be designed in such a way that it engages analysts in their work, rather than distract them. This quality has been characterized by Elmqvist et al. as fluid interaction [33], which describes interactions that allow analysts to be deeply engaged with their task, facilitated by the VA system. Fluid interaction brings the concept of "flow" from psychology [26] to the Information Visualization (InfoVis) world. Csikszentmihalyi defines flow as *"mental state of total immersion in an activity where the challenge of the activity and the skills of the participant are perfectly balanced, leading to high focus, involvement, and rewarding outcomes"* [26]. The qualities listed in that definition are obviously beneficial to solving complex analysis tasks in InfoVis and VA. Accordingly, Elmqvist et al. characterize fluid interaction around this concept as systems that show three properties [33], listed below together with brief explanations:

- **It promotes flow**: Interactions provide an "optimal experience" as defined above, rather than distracting, confusing, or annoying the analyst.
- **It supports direct manipulation**: Interactions take place as close to the representations of the data they affect, rather than at disconnected parts of the interface.
- **It minimizes the gulfs of action**: Interactions reduce the mismatch between analysts' mental model of the system state and the actual system state (gulf of evaluation) and it reduces the mismatch between the intentions for using the system and the actions allowed by the system (gulf of execution).

Providing fluid interactions has, since then been a goal in the design and implementation of visualizations and in turn also of VA systems.

Breaking the "flow" on the other hand has been shown to be detrimental, not only to the user experience but also to the successfulness

Response	Interaction goal
.1s	*Perceptual Update:* Computations initiated through direct interaction with the view.
1s	*Immediate Response:* Feedback on GUI interactions like tuning computational parameters.
10s	*Task completion:* Initiating a computational task, such as a query or complex filter operation on large datasets.

Table 1: Expected response times for different levels of responsiveness to maintain fluid interaction for different purposes, as compiled by Angelini et al. [6].

of the analysis. Tominski and Schumann identify three threats to fluid interaction: temporal separation, spatial separation, and conceptual separation [108, pp. 140]. Temporal separation describes a temporal delay in the interaction, spatial separation describes a distance in the place of the interaction and the effect it has, and conceptual separation describes a mismatch in the internal model of the system and analysts' comprehension of it.

PVA, the focus of this thesis, addresses the temporal separation, which can be caused by the ever-growing size of datasets and the increasing complexity of analysis methods, as both often increase computation times. Since interactions require a response from the system (often a rerun of parts of the analysis), this response becomes delayed, the more data is in the analysis pipeline.

The impact of such delay on human-computer interaction has been well studied. For example, in 1968 Miller was already able to show how for many computer-supported tasks, waiting up to ten seconds produced good results, while performance measurably declined for longer delays [79]. Shneiderman similarly found there to be a "sweet spot" (at around twelve seconds in his study) in response times for certain tasks, below and above which participants' error rates increased [106]. Accordingly, Card et al. identified three levels of delay that were acceptable for different tasks: below .1s for perceptual processing (*"maintaining the illusion of animation"* [18]), below 1s for immediate responses (*"serve[s] to keep the parties of the interaction informed that they are still engaged in an interaction"* [18]), and below 10s for unit tasks (*"completing an elementary task"* [18]). Recently, Liu and Heer studied the general effects of delay on the exploratory visual analysis, finding that even adding delay of 500ms had a noticeable impact on exploration strategies, and it decreased user activity and dataset coverage [72]. Angelini et al. [6] later compiled expected response times for different interactions in PVA (see Table 1).

These findings are clearly supporting the claims of fluid interaction, where a tight coupling of the interaction through low response rates is correlated with performance gains. The takeaway here is that in order for a VA system to provide fluid interactivity, it must keep response times to user input low.

2.1.3 *Reducing Interaction Latency with Sampling in VA*

One approach in non-progressive VA to reduce the latency is to run the computation on a smaller subset (a so-called "sample") of the data. The process for generating such a subset is also called sampling. Sampling can be a viable option, because approximate results can be just as useful as definitive results [67]. Reducing the size of the dataset can obviously increase the update rate in VA, as it reduces the complexity of the analysis. One standard heuristic for sampling is statistic representativeness, meaning that the value distributions in the sample are similar to those in the overall dataset. How this representativeness is defined, though, depends on the analysis scenario, as what makes the approximate results useful depends on what analysts want to do with it. This is further demonstrated in the examples below; more complete surveys can be found in the work by Olken and Rotem [86] and Cormode et al. [25].

A simple interpretation of representativeness is that the frequency with which certain values appear in the sample should correlate with the frequency of the full dataset. Accordingly, in the commonly used random uniform sampling without replacement, items from the full dataset are sampled with uniform probability, which means that values appearing across many items have a high probability of appearing in the sample. This sampling is often used in VA for overview tasks, as this heuristic generally performs well at maintaining frequent values. It does, however, have known shortcomings in maintaining rare values in small sample sizes, which can be challenging for tasks like training models on unbalanced class labels. Here, stratified sampling can be a solution, which facets the data along the class attribute and then randomly selects from within each facet a specified number of items. That way, it is possible to guarantee a certain number of values for each class in the sample.

The choice in a useful sampling method is also influenced by the underlying dataset. For example, for sampling network data containing links and nodes, the structure of the network is often relevant for the analysis. Again there are different ways of preserving a useful structure in different tasks. Wu et al. [120], for example, showed that three graph sampling methods preserve different visual features in a node-link diagram of the sample. Similarly, for geographic data, the spatial distribution of the data is essential for its analysis. Park et al., for example, present visualization-aware sampling that selects elements in a way that the visualization of geospatial data maximizes the visual fidelity at each zoom level [89]. In contrast, Zheng et al.

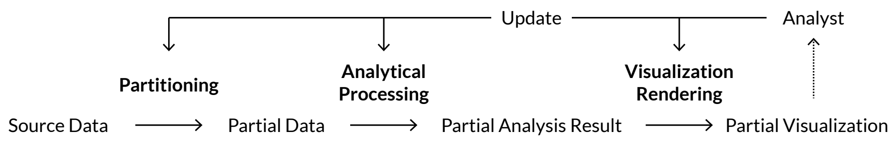


Figure 4: A basic pipeline of the PVA process, adapted from Li and Ma [68].

present a method that reduces the error in Kernel density estimates on geospatial data [129].

The takeaway here is that sampling is a successful approach for reducing the size of large data for reducing the computation time of the analysis. Rather than processing the entire dataset, the idea is to find a representative subset of the data on which to run the analytic computation. Which sampling method is useful depends on the analysis task, as well as the type of the dataset. These conclusions are relevant for the contributions in this thesis, in that sampling is an important mechanism for producing early, partial visualizations in PVA, and the same conditions (task- and data-dependency) hold true there as well.

2.2 PROGRESSIVE VISUAL ANALYTICS

Another response to reducing the delay in visualizing large datasets is Progressive Visual Analytics (PVA). The term PVA was introduced by Stolper et al. in 2014 [107], and has since been taken up by other researchers. In contrast to regular VA, partial visualizations show early results of long-running computations are shown to analysts, rather than forcing them to wait until the full computation completes. This means analysts can maintain their flow, as they get to see the results much earlier. In contrast to performing VA on sampled data, analysts also eventually get to see the entire dataset. Thus, the goal of PVA was characterized in a Dagstuhl report as *putting analysts back into the loop of long-running computations* [39]. Figure 4 depicts a simple model of the PVA process. There exist different interpretations of PVA [60, pp.11]. The remainder of this thesis uses the definition by Fekete and Primet, who define the progressive data analysis around a function with three properties [41]:

1. A constant computation time of partial computation steps.
2. A guaranteed computation time per step that lies below a certain threshold time.
3. Results that converge over time.

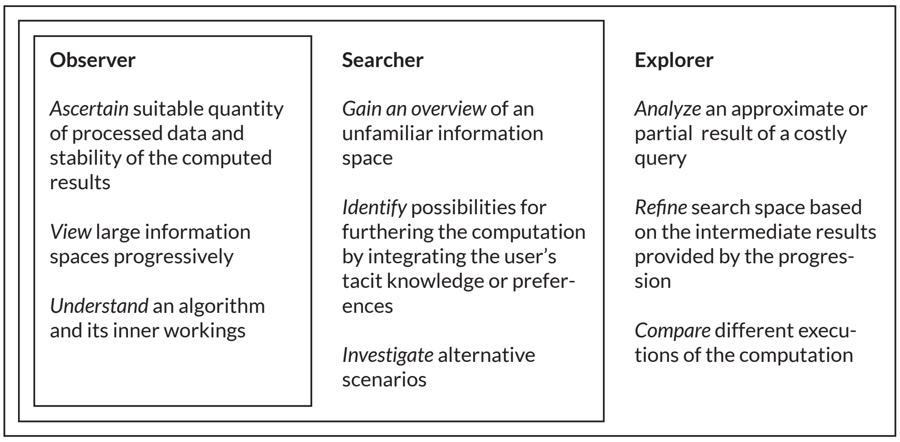
The PVA literature can generally be distinguished into two general approaches for producing partial visualizations: Either by iteratively outputting partial visualizations from computation steps, or by incrementally outputting partial visualizations from partitions of the data. Examples for iterative computations are online dynamic graph drawing [44] and dimensionality reduction algorithms like t-sne [93]. This thesis focuses on incremental PVA.

2.2.1 *Benefits of PVA*

There are many reasons for using PVA [6]. This section highlights some of these benefits to further motivate the contributions in this thesis.

ANALYSTS CAN MAINTAIN THEIR FLOW The main benefit of PVA used to motivate it in the Introduction Section 1.1 is that showing useful, early results allows analysts to maintain their analysis flow during long-running computations. This distinguishes it from non-progressive (also called monolithic) VA approaches, where they would need to wait for hours, days, or weeks, before getting to see the final result. As discussed in Section 2.1.2, flow is essential for making the interactive visual analysis successful. Beyond not forcing analysts to wait for the analysis to gain early insights, PVA also allows maintaining flow when adjusting the analysis. For example, as outlined by Micallef et al. [78], an analyst may take on a passive role in the analytic process at first, starting out by passively observing the progressive visualization. Then, once they notice that the visualization does not produce useful results, switch to a more active role and begin adjusting the parameters of the computation. Thus, analysts can switch roles within one PVA session as summarized in the hierarchical model depicted in Figure 5. Rather than restarting the entire computation, whenever results do not return what the analyst was looking for, they can switch gears to adjust the parameters of the progression "to help [it] along" [78]. While this aspect of PVA helps maintain the flow, it also makes the analysis more efficient, since analysts do not have to redo the entire computation to adjust the analytic computation, but can dynamically do so on the fly.

ANALYSTS CAN TERMINATE THE COMPUTATION EARLY Another benefit is that analysts cannot only maintain their flow based on early results, but they can also terminate the computation early, which saves time. Terminating the computation early may happen for multiple reasons. One such reason is that the partial visualizations were useful, and the analyst was able to complete their task based on partial visualizations; they found what they were looking for. Alternatively, the analyst might detect that the results are *not* useful (for example, due to a poor choice in algorithms), and decide to stop the computation to reconfigure the analysis. In both cases, they save time by terminating early compared to waiting until completion. Some recent studies have indeed confirmed that analysts in PVA often com-



Observer

Ascertain suitable quantity of processed data and stability of the computed results

View large information spaces progressively

Understand an algorithm and its inner workings

Searcher

Gain an overview of an unfamiliar information space

Identify possibilities for furthering the computation by integrating the user's tacit knowledge or preferences

Investigate alternative scenarios

Explorer

Analyze an approximate or partial result of a costly query

Refine search space based on the intermediate results provided by the progression

Compare different executions of the computation

Figure 5: User roles and tasks in PVA (reproduced with permission from [40, Ch.7.4]). Important to note here is that roles and tasks are hierarchical, because analysts in PVA can change their interest while the computation is ongoing.

plete their tasks earlier using progressive systems compared to non-progressive systems, while still generating correct insights [94, 127].

ANALYSTS CAN STEER THE PROGRESSION In PVA, analysts interact with an ongoing computation, rather than a finished result, which allows them to steer the computation based on their insights, while it is still running. Some literature uses the term steering inconsistently, describing analysts controlling parameters of the algorithm [10], prioritizing data subspaces [27], or prioritizing parts of the result [93]. In light of this "confusion", Raveneau compiled the following encompassing definition of steering in PVA:

> *"An analyst steers the algorithm in a Progressive Visual Analytics system, when 1/ they express a constraint on a subset of the data or of the results, and 2/ they demand that the algorithm satisfies the constraint during the next execution iteration(s), i.e., without restarting."* [98, p.84]

The contributions in this thesis use steering in terms of constraining the algorithm on a subset of the data. One aspect of steering under this definition is that it allows analysts to focus the computational resources on relevant subspaces, rather than "wasting" them on data irrelevant to their task. For example, while running a progressive analytic over a decade-worth of data, based on the early results the analyst may identify an interesting pattern in a particular week. Rather than processing data from the entire data space or restarting the computation to filter, the analyst in PVA can instead focus the computation on that interesting data on the go. They can thus complete the computation locally earlier, i.e., getting "the full picture" for the region of interest. In non-progressive VA, they would have had to wait until the full computation is done, to then explore the data, find the interesting pattern, and only then could investigate it. The added benefit of computational steering in PVA is that it allows analysts to confirm or reject hypotheses about the partial data much quicker, as they can directly engage with the computation. For example, they might find after focusing the computation on the interesting pattern, that that region turns out to be uninteresting when looking at the "full picture". The analyst can thus move onto the next interesting pattern, without interruption or restart. This further increases the efficiency of the progressive analysis, as reflected in experiments by Procopio et al., who showed that even non-expert users were able to use steering in their work [94].

2.2.2 *Analytical Computations in PVA*

A defining difference to analytical computations that derive models from data in non-progressive VA is that in PVA, only parts of the dataset can be accessed. In PVA, for which the partial results should converge over time, the underlying assumption is that the analytical computation produces the same results for parts of the data, as it does on the entire dataset. Some computations inherently fulfill this requirement: For example, when subtracting a static value from each entry in the data, for each item, this result is the same regardless of the partitioning. This is because these computations only look at the data element by element. Computations that relate elements to other elements, however, need to be adjusted to work progressively. For example, when computing outliers in the data (i.e., items in the data that significantly differ from all other items), the decision whether an item is considered an outlier depends on the other items in the chunk. Therefore, using this type of computation progressively inherently produces errors. A general challenge that the literature on progressive analytical computations addresses is thus to find ways of reducing that error, in order to use these computations progressively.

A related field is online (or incremental) learning, which explores ways for reducing the error on unsupervised learning algorithms discussed for VA [76]. Following Fekete and Primet, online learning algorithms can provide partial results much fast than running computations exhaustively, and they can be adapted to PVA by ensuring that results are produced within the human latency constraints [41]. This notion clearly highlights the importance of online learning for PVA. The scientific literature on online learning is abundant, in fact, there are diverse approaches looking at the problem of incremental clustering alone [3, 11, 55, 124]. As such, online learning is already widely used in PVA systems, for example Turkay et al. use an incremental variant of Principal Component Analysis (PCA) [110], and Fekete et al. have applied an incremental variant of k-means [37, 41] to produce partial visualizations of clusterings. Others have explored aspects of progressive analytical computations directly from a PVA perspective, developing progressive algorithms. Jo et al. for example first presented an algorithm for progressively computing kd-trees [62], which was then used as basis for the implementation of progressive k-nearest neighbors [59], and progressive Uniform Manifold Approximation (UMAP) [65]. Other progressive dimensionality reduction algorithms are progressive t stochastic neighbors encoding (t-sne) [93] and progressive Multi-dimensional Scaling (MDS) [118].

To assess the quality of the partial results of online algorithms, robust statistics can be applied. Robust here signifies that *"the shape of the true underlying distribution deviates slightly from the assumed model"* [58], meaning that the approximate, partial result resembles the complete result of the computation. This research field is highly-relevant in PVA, where analysts work with partial (i.e., assumed) results, requiring them to perform their analysis under the uncertainty that these results may change. By supplying them with robust statistics, analysts can be assured that the result after the computation completes lies within a certain error bound of the partial result. One central challenge is that ensuring robust partial results generally requires large sample sizes, which can stand in contrast to PVA's requirements of the human latency constraints. Pébay, for instance, states the well known conflict between online learning and robust statistics, in that *"algorithms for calculating [robust statistics] for the sake of execution speed [...] lead to unacceptable numerical instability"* [91]. In response to this, dedicated online robust statistics have been proposed, for example, in Pébay's work on higher moments [91] or for Welford's method for variance [9].

2.2.3 *Partial visualizations in PVA*

Just as with visualization in VA, its purpose in PVA is to encode the results of the analytic computation. The main difference compared to non-progressive VA is that the results in the visualization are inherently incomplete and that the visualization continuously changes with every new chunk of data. This section looks at how the literature addresses this difference.

One way the literature addresses changes in the visualization is to inform the user about the result incompleteness upfront. A common example of this are error bands on bar charts that show the confidence interval [61, 94]. Turkay et al. show a method that adjusts the size of bins in a binned scatterplot to encode the error, in that the larger the bins, the greater the chance that the location of dots in these regions may change at a later stage, as more and more data is loaded [110].

An alternative approach is to inform the user retroactively about changes in the visualization, since they last looked at it. Moritz et al.

propose so-called optimistic visualization [80]. The general approach is to provide analysts with fast approximate results that are refined as analysts work with the data. Later, when the computation is completed, analysts can look back at their earlier results and are informed about in what areas and by how much the initial approximation differed from the final result. Then, they can choose, what parts of the analysis to rerun based on these "corrections". A related approach to optimistic visualization is ProReveal [61]. Here, analysts can formulate hypotheses on progressive visualization by specifying the range in which they expect the final result to lie. Whenever the progressive value exceeds these bands, the analyst is notified, so they can reconsider their insights.

A third option found in the literature is to design the visualization in such a way that it avoids these changes in the first place. Waterink et al. present a technique to reduce the visual fluctuation in progressive visualization between two steps, which they name "popping artifacts" [117]. Chen et al. look at the considerations for rendering scatterplots and dot map visualizations, presenting adjustments to the sampling step of the PVA pipeline to achieve a more suitable visualization [23]. Rosenbaum and Schumann [99], on the other hand, explore a technique for prioritizing the most salient features of the visualization, before refining the details by adjusting the rendering step of the PVA pipeline. Therefore, the overall layout of the data remains stable, and only changes in the details. While their approach does not strictly fall under the definition of PVA, since this technique requires knowledge about the final visualization before it can be applied — the approach was proposed to overcome limitations of limited bandwidth — it shows potential beyond just plotting more and more data as it arrives. Another related example are Progressive Parallel Coordinates [100], a progressive visualization technique for high/dimensional data. Therein, the authors demonstrate how the same visualization technique can be progressively visualized in different ways, each tailored to a particular user interest.

2.2.4 *Interaction in PVA*

Earlier, this chapter noted that a goal of PVA is to put the human back in the loop in long-running computations [39]. A conceptual difference to non-progressive VA from an interaction perspective is that analysts no longer operate on a finished analytic result, but instead with an ongoing analytic process. However, leveraging the benefits of PVA — actually putting the human back in the loop — requires that there is no *noticeable* difference between the two interactions from the analyst's point of view, in that interaction on a process should *behave* exactly like interaction on the final result [6]. Thus, interactions with the analytic process must be tightly coupled with interactions with the progression. Mühlbacher et al. distinguish between two degrees of control that analysts gain by interacting with an ongoing computation: execution control (controlling the input of the computation) and result control (controlling its output) [82]. This section looks at how the related work addresses interaction in terms of these two aspects.

Mühlbacher et al. divide execution control into cancellation and prioritization. Cancellation is an often cited benefit of PVA, and it is often used as a main argument for applying progressive visualization [43, 94, 127]. Cancellation is also discussed as it facilitates rapidly restarting the computation for fine-tuning parameters of the computation [10]. The related has also looked at coupling view interaction with prioritization (regarding remaining parts of the computation). In terms of the interaction coupling in PVA, interaction on the visualization affects data that is already in the visualization as well as data that in the future will appear in that part of the visualization the analyst interacted with. The goal in PVA literature is thus to find ways of prioritizing this "invisible" data. An example of prioritization was presented by Williams and Munzner, who discuss a method for prioritizing data in MDS, based on a user-selected bin [118]. Here, the interaction was fully decoupled from view space exploration and purely served to interact with the progression. The Sherpa method by Cui et al., on the other hand, steers analytic computations over sequential data such as genomes or time series, based on brushing interactions that analysts perform for exploring the data [27]. The interaction with the progression in this case is tightly coupled with the view space interaction, and participants in the evaluation even stated that they had intuitively expected this behavior. Pezzotti et al. present steering for a t-sne layout, which couples view space interactions like brushing and magic lenses with the prioritization [93], again showing a tight coupling. In terms of the coupling of interaction, the related work provides dedicated solutions for coupling control over the progression with prioritization.

Regarding result control, prior work has mostly discussed ways in which to interact with the progression in PVA, i.e., inner result control in Mühlbacher et al.'s terms [82], which allows for controlling the output within one analysis session. Badam et al., for example, explicitly look at the challenge of "steering the craft" [10]. In terms of the result control, they present a system that uses widgets to play, pause, and stop the progression, a progress bar that allows "going back in time" to view the progression at earlier stages, as well as buttons and sliders to set specific parameters of the underlying computation. Turkay et al. discuss interaction considerations for PVA on high-dimensional data, and similarly use widgets to control the progression, including widgets for manually restarting the computation and setting the chunk size [110]. Looking at coupling between the view and computation, the related work has mostly separated result control from view, i.e., the interaction is “spatially separated” [108, pp. 140]. Outer result control, on the other hand, that is, configuring the PVA process for multiple consecutive executions has only been addressed in terms of toolkits, which allow building and configuring PVA processes. P5 [68] for example proposes a declarative approach for configuring the progression. ProgressiVis [37] addresses the architectural challenges of progressive data structures and modules. These frameworks are, however, still under development, and mostly used experimentally.

2.3 SUMMARY

This chapter provided the theoretical background on related concepts in VA and its specialization PVA by exploring the scientific literature. Based on this, the addressed research challenges and provided contributions in this thesis are motivated in the following chapter.

3 IMPROVING THE USEFULNESS OF PARTIAL VISUALIZATIONS IN PROGRESSIVE VISUAL ANALYTICS

This chapter characterizes "usefulness" of partial visualizations in PVA and provides an overview of the contributions in this thesis towards the research challenges.

3.1 CHARACTERIZING USEFULNESS OF PARTIAL VISUALIZATIONS IN PROGRESSIVE VISUAL ANALYTICS

The interpretation of usefulness for partial visualizations in PVA in the scientific literature has evolved over time. In the work by Stolper et al., which introduced Progressive Visual Analytics (PVA), usefulness referred to partial results that were "semantically meaningful" to analysts [107]. In a following paper by Fekete and Primet the interpretation of usefulness was further refined, suggesting for the visualization to converge to the final result "as quickly as possible" [41]. The authors, however, also note that guaranteeing such a quantitative usefulness is challenging, as it depends on many factors, including the analysis function, its parameters, the data, the computation method, and the way in which this convergence is measured.

Later work emphasized the importance of subjective interpretations of usefulness. In particular, the work by Moritz et al. on optimistic visualization [80] and the work by Jo et al. on progressive guards [61] emphasize that the partial visualization must be "trustworthy", in order for analysts to continue their analysis. The idea behind this perspective is clear, as only when analysts trust the visualization can they move on in their analysis, and thus *actually make use* of the partial visualization. A similar notion can be found in the review of PVA literature by Angelini et al., who explicitly note that *"a partial result is only useful [...] if it shows what the analyst needs to see"* [6]. This quote clearly brings out the idea that usefulness can only be viewed in connection with a particular user interest in a particular scenario.

The review further qualifies the usefulness of the partial results at four levels as "meaningful" at first, "trustworthy" later on, and then "significant" [6] and finally, of course, "complete" once the progression concludes. These levels are depicted in Figure 6. It also provides recommendations for supporting analysts through partial results. To support the first level — that is *meaningful* partial results — the system needs to ensure immediacy, significance, and actionabil-

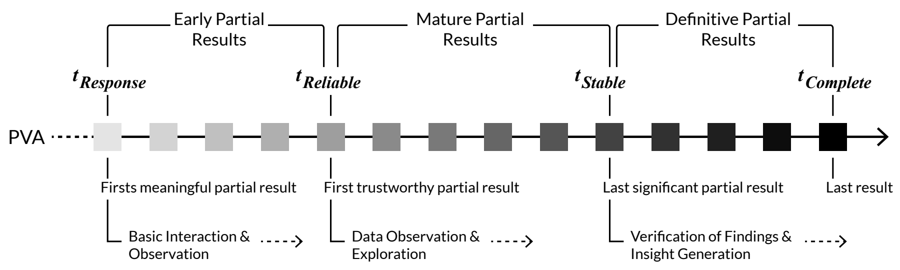
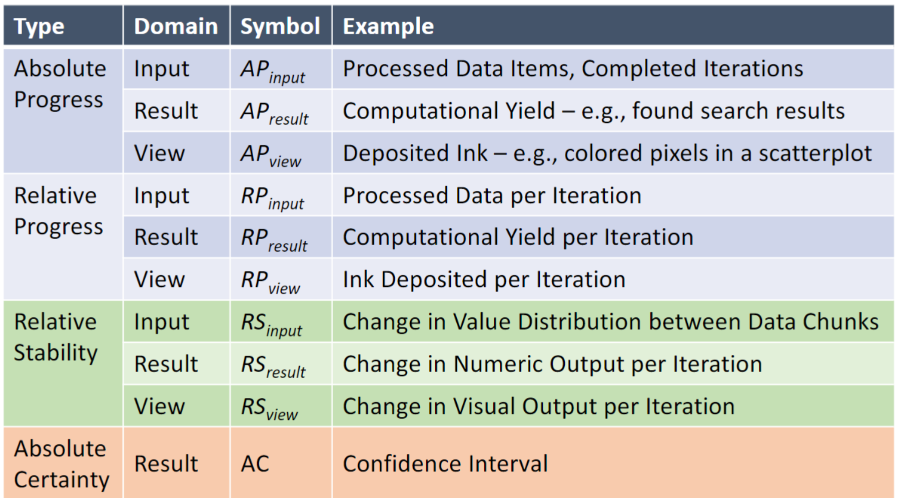

Early Partial Results
Mature Partial Results
Definitive Partial Results
t Response
t Reliable
t Stable
t Complete
PVA
Firsts meaningful partial result
First trustworthy partial result
Last significant partial result
Last result
Basic Interaction & Observation
Data Observation & Exploration
Verification of Findings & Insight Generation

Figure 6: Timeline conceptually showing the increasing usefulness in partial visualizations for PVA as presented by Angelini et al.. While analysts in non-progressive VA would have to wait until the entire computation is completed in order to make use of it, the partial visualization in PVA becomes useful much earlier (reproduced from [6]).

ity. To support the analysis at the trustworthy level, these criteria are extended towards *"establishing trust in the still incomplete results"* [6], meaning that the system now needs to communicate the uncertainty of the ongoing process to analysts. That is, the system needs to inform analysts how close or far away the partial result is from a stable, significant result, both in terms of the data (such as confidence bounds), but also about the process (such as provenance information) [6]. To support the third level of usefulness, where partial results remain mostly stable as no further significant changes are expected, PVA systems should also include ways of assessing the stability of results, by informing the judgement of the state of the process and the progress of the progress [6]. One aspect to note here is that the notion of usefulness in general and in these recommendations in particular is based on qualitative terms rather than quantitative metrics, indicating that analysts need to subjectively assess usefulness.

In a follow-up work, the authors then defined ten quantitative descriptive indicators for the progression to support analysts in their assessment. These measure the quality of input, result, and view in terms of progress, stability, and certainty [5] (see Figure 7). Visualizing these indicators in an interface allows analysts to subjectively assess the usefulness of the partial visualization, meaning that they can now — based on numeric values — determine, where on the meaningful–trustworthy–significant spectrum the partial visualization currently is. This intuitively makes sense, in that the same partial result can be useful in some scenarios, but it may not suffice in others, and indicator metrics help inform analysts' decision on whether it is the case.

Relying on the subjective assessment of usefulness naturally opens the door for cognitive biases in PVA. Biases are a systematic and involuntary way of how humans deviate from rational judgement, regardless of intelligence and domain expertise [78]. These are widely understood as a general challenge to the decision-making process in



Type	Domain	Symbol	Example
Absolute Progress	Input	AP_{input}	Processed Data Items, Completed Iterations
	Result	AP_{result}	Computational Yield – e.g., found search results
	View	AP_{view}	Deposited Ink – e.g., colored pixels in a scatterplot
Relative Progress	Input	RP_{input}	Processed Data per Iteration
	Result	RP_{result}	Computational Yield per Iteration
	View	RP_{view}	Ink Deposited per Iteration
Relative Stability	Input	RS_{input}	Change in Value Distribution between Data Chunks
	Result	RS_{result}	Change in Numeric Output per Iteration
	View	RS_{view}	Change in Visual Output per Iteration
Absolute Certainty	Result	AC	Confidence Interval

Figure 7: Ten quality indicators for PVA (reproduced from Angelini et al. [5]).

VA [114], and by showing partial results, PVA in essence makes an already uncertainty-laden process even more prone to biases.

Micallef et al. identify uncertainty bias, illusion bias, control bias, and anchoring bias as additional potential pitfalls when working with progressive visualizations [78]. Procopio et al. later studied the impact of each of these biases on the analysis [94] in a series of four crowdsourced user studies on Mechanical Turk. They found that, while progressive visualization led to time savings (around 88% in their case), it could also lead to a measurable reduction in accuracy, indicative of the presence of these biases. Yet, even under the presence of uncertainty, even non-expert analysts were able to perform their tasks. This was also shown in other work by Patil et al., who studied the effect of four different encodings of uncertainty on bar charts [90]. While they did not find either of these encodings to be superior, they generally found that users were able to successfully perform tasks under the uncertainty introduced by the progressive visualization. Some work has also shown how analysts in PVA engage with uncertainty in practice to alleviate their biases. Zgraggen et al., for example, quote analysts who purposefully decided to wait longer to make sure that patterns they observe remain after more data is processed [127]. These user studies suggest that appropriately incorporating quality metrics in the interface can nevertheless make an uncertain visualization useful.

The takeaway from this characterization here is that usefulness of partial visualizations in PVA literature refers to how well a visualization supports analysts in completing their task. Usefulness itself cannot be directly measured quantitatively, but requires that analysts assess it subjectively in the context of their specific analysis scenario. PVA systems can support analysts in their assessment, for example, through suitable encodings and metrics, which help reduce the impact of cognitive biases on the assessment. At the same time, improving the usefulness can only be achieved by allowing the analyst that actually uses a PVA system to configure it to their subjective needs. This, of course, requires that the system is in fact configurable. Moreover, as the partial visualization becomes increasingly complete throughout the progression, analysts interests — and, therefore, their assessment of usefulness — can shift, which requires that the system must be dynamically configurable.

This theoretical goal stands in clear contrast to how many PVA systems are currently designed in practice, in that they often provide hardwired solutions tailored to specific analysis scenarios, rather than being flexible and tailorable. This is, of course, completely understandable, given that PVA as a research field is still relatively young with many specific research challenges unanswered, but it limits usefulness to these scenarios. Nevertheless, partial visualizations in systems built on these solutions inherit their limitations and, thereby, limit their usefulness.

3.2 PROPOSING A CRITERIA-BASED DELINEATION OF USEFULNESS OF PARTIAL VISUALIZATIONS IN PVA

To reduce the gap between theory and practice outlined above, this thesis makes contributions towards improving the usefulness of partial visualizations in PVA by giving analysts the ability to configure the underlying PVA process to their interest.

As shown in the characterization above, quality metrics are currently used to *assess* the usefulness of partial visualizations based on a quantitative value in terms of the stages proposed by Angelini et al. [6]. This thesis expands on this notion by proposing means of using well-established quality criteria from VA literature to *improve* the usefulness based on analysts' subjective perspectives. This results in two interpretations that can be applied independently of each other, with metrics enabling an assessment based on quantitative values, and the criteria providing qualitative support to improve it.

Based on earlier work, for example by Mackinlay [77], Card [19, p.558], and Don Norman [84, p.97], Tominski and Schumann formulate three quality criteria for visualization: Expressiveness (usefulness from a data perspective), effectiveness (usefulness from a task perspective), and efficiency (usefulness from a resource perspective) [108, pp.17]. These criteria formulate how successful the visualization is in supporting analysts in completing their tasks. This makes them a good fit for improving usefulness of partial visualizations as well, which in the words of Angelini et al. are *"only useful [...] if it shows what the analyst needs to see"* [6]. The idea is, in short, to allow analysts to improve the usefulness of partial visualizations by increasing their expressiveness, effectiveness, and efficiency.

There are many ways in which analysts can configure the PVA process to influence these criteria (parameters, the analytic computation, the visual encoding, . . .). This thesis focuses on contributions to controlling the incoming data to be visualized as the central point of control, inspired by Tufte's principle of good statistical graphics: *"Above all else show the data"* [109, p.92]. Having control over the data is arguably the most influential "set screw" on the usefulness of the visualization. In the remainder of this chapter, the three quality criteria are characterized from the perspective of controlling the input data of partial visualizations, and the contributions towards improving each are outlined.

3.3 IMPROVING THE EXPRESSIVENESS OF PARTIAL VISUALIZATIONS WITH TAILORABLE SAMPLING

Tominski and Schumann characterize a visualization as expressive, *"if it communicates the desired information in the data, and only this information"* [108, p.17] and that it *"objectively reflects the information we need to accomplish our task"* [108, p.17]. Expressiveness, thus, can be applied to improve usefulness along a data perspective. When considering expressiveness under the PVA lens, the obvious difference to non-progressive VA is that not all data is available at once, and so it may not even be possible to actually visualize that desired information. The mechanism that controls the general order of the data in the PVA process is called data partitioning [68], and the standard method for partitioning the data is sampling. To improve the expressiveness of partial visualizations, analysts in PVA should tailor the sampling, such that the data holding the information needed to accomplish the task is (in the words of Tominski and Schumann) "objectively reflected" as early as possible.

This is reflected in the literature through the following requirements [6, p.23]:

- Employ an adaptive sampling mechanism (convergence & temporal constraints) [110]
- Allow efficient and deterministic query processing over progressive samples, without the system itself trying to reason about specific sampling strategies or confident estimation [21]

3.3.1 *Research Background*

The de facto default sampling method in PVA remains random uniform sampling without replacement [10, 61, 68, 110, 127], which supports overview tasks best. There also exist sampling approaches that allow adjusting the data stream for specific user interests. One example is pyramid-based sampling, which considers density- and outlier-preservation, as well as temporal coherence between chunks as representative characteristics for progressive scatter plots [23]. Pyramid sampling achieves this by selecting those data points that will change the visualization most noticeably, compared to data points that would just overplot existing data. Other related approaches can be found in the database community, in particular the field of approximate query processing, where one challenge is to speed up analytic queries over large, multivariate datasets, while maintaining the validity of the results. Li et al. present the Sampling Cube algorithm [69], which automatically expands the data sample, whenever the confidence intervals of a computation would exceed a certain limit. That way, the validity of results can be kept consistent, by only processing as much data as possible.

Other approaches sample streaming data, which similar to the data in PVA is made available in partitions, yet in contrast to PVA is usually analyzed in windows, i.e., only the most recent data is considered relevant. The most widely used method is reservoir sampling [113], which builds up a random sample progressively from all data processed up to a particular point in time. While it discards data no longer considered as relevant — which makes it unsuitable as a partitioning method in PVA — reservoir sampling has nevertheless been used to support progressive analysis settings. For example, reservoir sampling can be used for enabling progressive indexes, which increase query performance on data that is not fully available at once. Recent work by Hohenstein shows that a slight adjustment can be made to this progressive indexing algorithm [56] to overcome sorting biases in the partial data through reservoir sampling, making it fit for overview tasks in PVA [54].

Overall, however, existing PVA-samplings are hard-wired to address a specific analysis scenario, and tailoring them is generally not a consideration — let alone while the computation is running. This clearly limits analysts' control over the incoming data stream, meaning that the partial visualization may not be able to represent the information needed to accomplish the task.

3.3.2 *Research Challenge*

The research challenge here is that a new approach is needed for that provides analysts with tailorable sampling, requiring conceptual and technical contributions. Since the user interest differs between scenarios, this new approach not only needs to be tailorable by analysts *before* the analysis starts, but since that interest can also change *during* the analysis, it needs to be tailorable without requiring a restart.

3.3.3 *Contribution*

To address this challenge, this thesis contributes tailorable sampling for PVA. Tailorable sampling standardizes the sampling process along three steps in a pipeline, such that this process becomes tailorable by exchanging the operators used at each step. In contrast to approaches for non-progressive VA where sampling is an operation performed *once* before the analysis, tailorable sampling for PVA considers sampling as a process that runs in parallel to the analysis, with both influencing each other: Based on the insights gained from the samples, analysts can complete their task, and when analysts change their task during the analysis, the sampling can be tailored to reflect this. This means that the expressiveness of partial visualizations can be improved throughout the progressive analysis. The manuscript included in Chapter 6 presents tailorable sampling with a pipeline and evaluates its modularity and versatility in a series of use cases.

3.4 IMPROVING THE EFFECTIVENESS OF PARTIAL VISUALIZATIONS WITH DOI FUNCTIONS

Tominski and Schumann characterize a visualization as effective, *"if it is geared to the human sensory and motor systems, that is, our abilities to observe and interact with our environment"* [108, p.17]. Effectiveness, thus, *"captures how well we can extract the information needed for our task from a visual representation"* [108, p.17] and provides a task perspective on improving usefulness. In the PVA context, an added challenge is that the "information needed for the task" may not be available, or, as time goes on and more data is processed, even if interesting data is available, it may be obfuscated among all other data. Therefore, to improve the effectiveness of partial visualizations, analysts in PVA should ensure that the visual representation allows that interesting information needed for their tasks can be extracted.

This is reflected in the literature through the following requirements [6, p.23]:

- Support the interpretation of the evolution of the results through suitable visualizations [110]
- Use consistently visualized quality measures [10]
- Managing the partial results in the visual interface should not interfere with the user's cognitive workflow [107]

3.4.1 *Research Background*

Optimizing the inherent visual complexity from representing millions of points through visualization is an inherent challenge for VA, as screen real estate is limited. Keim's pixel-based visualizations [63] approach this by minimizing the size of visual elements that represent each data point to the size of a single pixel, but even then, the visualization is limited by the screen resolution, but also overplotting quickly becomes a challenge. Another way of reducing the visual complexity in VA is to assume that not all data is equally important to the analyst, and to focus the analysis on interesting data and an approach for ranking the data by its importance are so-called Degree of Interest (DOI) functions. Furnas first proposed DOI functions for visualization as early as 1981 [45]. In this early work on graph visualization, per-node importance is computed from an inherent (apriori) interest like the value of an attribute, and a dynamic (posteriori) interest that changes as analysts interact with the data, for example the distance to a focused node in a graph. This allows showing parts of the graph to the user that are far apart, yet interesting to their current focus point.

DOI functions have since been widely applied in VA across different tasks and data types [1, 50, 135]. Elmqvist and Fekete, for example, apply the principle of DOI function for filling a visual entity budget [32]. The idea is to assign a budget based on a metric like rendering time that describes how many items can be rendered. A DOI function can be then be used to decide, which points to render within that budget, selecting the most interesting items first until the budget is exhausted. DOI functions are, thus, a candidate for making partial visualizations more useful, since they allow emphasizing those parts of the data that are most relevant to users. In particular, since the size of even the partial dataset can quickly become massive, having a DOI function could help prioritize interesting data in the rendering.

However, transferring DOI functions to PVA can be non-trivial. This is because the "context" in which user interest is measured changes over time, as more and more data gets processed. This could mean, for example, that parts of the data change their interest scores over time: An item that is of great interest in context with very little data at early stages of the progression may become uninteresting, once more data has been processed, or vice-versa. In other words, interest scores computed previously need to be adjusted to the context of incoming data, and interest scores computed on incoming data need to be adjusted to the context of previous scores. To overcome this, we would essentially need to recompute all interest scores every time new data arrives in the visualization (and every time users perform an interaction). As a result of this, DOI functions are only rarely used in PVA. This is a clear limitation of PVA, as partial visualizations of large datasets are a clear candidate for DOI-based improvements.

3.4.2 *Research Challenge*

The research challenge, thus, lies in providing analysts with means to reduce the runtime of DOI functions by reducing their input to a suitable subset of the data, requiring conceptual and technical contributions. The difficulty here is that what is considered *suitable* requires analysts' subjective input: The subset should on one hand include as many data items as possible to maintain valid interest values, but on the other hand only include as few items as necessary to reduce the computation time. In other words, a solution needs to be flexible to account for different analysis scenarios.

3.4.3 *Contribution*

To address this challenge, this thesis contributes a catalog of strategies for reducing the cost of recomputing the DOI functions. This is done in two ways. First, rather than updating the interest value of all data processed so far with each new incoming chunk, strategies are proposed for limiting updates on items for which the interest most likely changed. Secondly, rather than measuring the interest value of the new data in context of all data, strategies are proposed for identifying a small, but representative subset. The paper in Chapter 7 details these strategies and evaluates them in a series of benchmarks, with results suggesting that there does not exist one "silver bullet" combination of strategies, but that strategies instead should be chosen per analysis scenario for the best results.

3.5 IMPROVING THE EFFICIENCY OF PARTIAL VISUALIZATIONS WITH STEERING

Tominski and Schumann characterize efficiency as follows: *"The gains from using an interactive visual approach should outweigh the computational resources and human effort needed to carry out the analysis"* [108, p.18]. Efficiency, thus, can be applied to improve usefulness along a resource perspective. To make a visualization more efficient, thus, means either reducing its cost or increasing its gains. Its added costs for actually using partial visualization is a general challenge for PVA, as tool support is generally sparse (requiring additional implementation effort), and basically all steps along the VA pipeline need to be reconsidered when things are to be used progressively (requiring additional mental effort). At the same time, reducing human and computational efforts by showing partial results is a central motivation for using PVA in the first place. Giving analysts control over the data stream through steering is a major factor in making this trade-off. Therefore, to improve the efficiency of partial visualizations, analysts in PVA should steer the progression towards data of interest, to reduce the computational resources and human effort needed to carry out the analysis.

This is reflected in the literature through the following requirements [6, p.23]:

- Process interesting data early, so users can get satisfactory results quickly, halt processing, and move on to their next request [51]
- Allow users to focus the algorithm to subspaces of interest [107]
- Allow users to ignore irrelevant subspaces [107]
- Allow altering the sequence of intermediate results through prioritization [82]
- Provide inner result control for steering a single ongoing computation before it eventually returns a final result [82]

RESEARCH BACKGROUND Computational steering can be major advantage of PVA over non-progressive VA (see Figure 2.2.1), as it allows analysts to focus computational resources on a particular subspace of interest, thus making the visualization more efficient. However, existing steering approaches pose certain restrictions on the analysis scenario, which means that steering cannot always be applied.

An early example for steering in PVA from 2004 was proposed by Williams and Munzner, who use it to steer the dimensionality reduction method Multi-dimensional Scaling (MDS) [118]. MDS is often used to project high-dimensional data into two dimensions, by maintaining the relative Euclidean distances from high-dimensional data in the projection. It provides a "human-readable" representation for highly complex data by using the distance-similarity metaphor [36] in that similar items are located closer together in the projection than dissimilar items. The steering approach by Williams and Munzner allows analysts to prioritize a data subspace in a progressive variant of MDS, such that items from a selected bin are retrieved, first (see Figure 8). To achieve this, the approach maintains an index structure over remaining items to continuously rebin those and to then prioritize

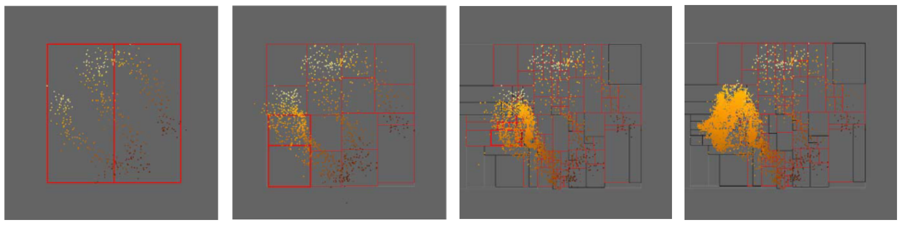


Figure 8: The steering mechanism presented by Williams and Munzner, prioritizing data for progressive dimensionality reduction using continuous rebinning (reproduced from Williams and Munzner [118]). Notice how bins (depicted as red rectangles) change throughout the progression, indicating continuous updates over the remaining data.

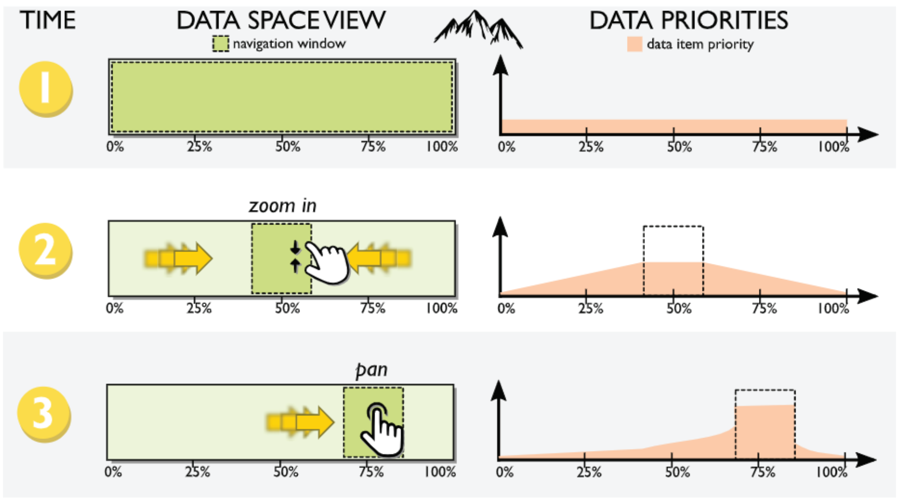


Figure 9: The Sherpa steering mechanism, allowing to prioritize data from selected intervals in the progression (reproduced from Cui et al. [27]). Interesting to note is the continuous "decay" of the prioritization function after the analyst shifts their attention to a different region between (2) and (3).

items from the selected bin. This rebinning step is the clear limitation of this approach, in that it becomes too costly on the large datasets often used in PVA, where looking at each data item is unfeasible.

Another approach called Sherpa was proposed by Cui et al. [27]. Sherpa allows analysts to steer the computation by prioritizing items from intervals of interest, which they can select by brushing an axis in their plots (see Figure 9). The approach then applies a boolean filter to the data retrieval, limiting it to elements from that selected interval. Sherpa, thus, does not require running expensive computations over all data, but simply utilizing the 1-to-1 mapping from selection in view space to an interval in the data. This is, however, a strict requirement for using Sherpa, as it does not support any visualization not relying on such a 1-to-1 mapping. For example, Sherpa does not support the MDS-based visualization used in the work by Williams and Munzer above, as the layout here is computed from multiple dimensions.

Yet, the inherent complexity and prerequisites of existing steering mechanisms in PVA currently pose a limitation on the *efficiency* of the partial visualization in cases where neither approach applies.

3.5.1 *Research Challenge*

To allow analysts to steer the computation whenever neither of the existing steering mechanism applies, a new steering mechanism is needed. The research challenge here is that this new approach must pose as few limitations on the analysis scenario as possible, allowing analysts to steer towards relevant subspaces across visualization techniques, while also only using limited resources, requiring conceptual and technical contributions.

CONTRIBUTION To address this challenge, this thesis contributes a steering mechanism that poses only minimal constraints on the analysis scenario. Concretely, the steering-by-example approach presented only requires analysts to specify a small set of interesting data items in order to steer the progression towards similar, yet unprocessed items. Steering-by-example works by training a decision tree classifier on the selected set of interesting items to identify those aspects that distinguish this selection from the rest of the data. The decision rules from that classifier are then combined into a boolean filter that can be added to the query that retrieves the next chunk, thus prioritizing similar data. This makes steering-by-example more flexible regarding supported data types and visualization techniques than existing approaches. The paper in Chapter 8 presents steering-by-example and evaluates it together with state-of-the-art steering methods across different use cases. Results indicate how this approach can be applied to increase the efficiency of partial visualizations across analysis scenarios.

3.6 SUMMARY

This chapter outlined the thesis' contributions towards improving the usefulness of partial visualizations in PVA. First, the meaning behind the term usefulness itself was characterized by reflecting on the related scientific literature, finding that usefulness is a subjective and scenario-specific quality that analysts decide on based on the quality metrics. Then, means were proposed for increasing this subjective usefulness of partial visualizations along three established quality criteria in VA: expressiveness, effectiveness, and efficiency. The papers detailing these contributions are found in the second part of this thesis.

4 DISCUSSION AND FUTURE WORK

This chapter summarizes the thesis contributions, provides reflections on the research questions, and explores future research directions.

4.1 CONTEXTUALIZING THE THESIS CONTRIBUTIONS

Before considering how each addresses the research questions, this section puts the contributions in context with each other, considering them along the PVA process. Li and Ma derive a process model for PVA from the conventional visualization pipeline, describing it as a pipeline consisting of three transformations (partitioning, analytical processing, and visualization rendering), which can be dynamically updated based on user interactions [68].

In this model, tailorable sampling can be clearly situated at the data partitioning step, while enabling DOI functions affects the visualization rendering. Steering-by-example on the other hand can be positioned in the interactive update of the partitioning step. Each contribution, thus, adjusts a different part of the process. Figure 10 provides an overview on the process perspective. Below, some synergies between the contributions are explored.

Steering-by-example and tailorable sampling both affect the data partitioning, and can be used in concert with each other. The two can influence each other, if analysts use sampling for getting an overview and steering for exploring details, as outlined in the work on tailorable sampling. In terms of the data stream, sampling defines a "rough" general prioritization of data items based on an initial user interest, and steering allows refining this order on-demand to explore detailed patterns of interest that arise in the partial visualization. Furthermore, the two can also influence each other vice-versa. As analysts investigate patterns of interest using steering, they gain new insights and may in turn adjust their tasks, which affects the tailoring

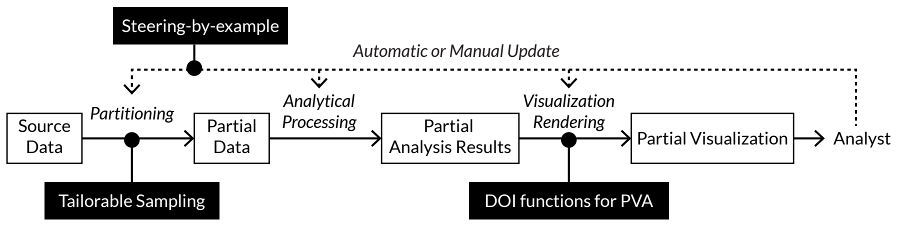


Figure 10: Putting the contributions of this thesis in context along the PVA pipeline adapted from Li and Ma [68].

of the sampling. The two contributions, thus, benefit the progressive explorer role described by Micallef et al., which describes analysts whose main interest in PVA "is for its flexible, steerable nature that allows adjusting an underlying computational process while it is running" [78].

Another potential for symbiosis exists between DOI functions and steering. The steering-by-example approach requires analysts to supply a set of interesting data items as input. DOI functions on the other hand can identify those items automatically, which means that the output of an appropriate DOI function could be used to automatically steer the progression towards interesting data. This, of course, requires that the DOI function accurately captures the user interest, in that the items identified as interesting are actually relevant to completing the analysis task. Otherwise, combining the two approaches to automate steering could potentially steer the analysis *away* from the task, thereby reducing the effectiveness of the partial visualization. Yet, there may be still be merit in this, automatically providing analysts with data they do *not* expect to broaden their view on the data. Actionable insights regarding the implications and implementation of this symbiosis, however, requires future research in this regard.

A third connection can be drawn between the papers on tailorable sampling and DOI functions in PVA, in that both define an order over the data by some interest-driven metric. One potential way of combining the two is to use a DOI function in the linearization step of the sampling pipeline. In terms of Furnas [45], the linearization step sorts the data based on apriori interest, which remains static throughout the analysis. Given that DOI functions can also capture posteriori interest based on user interactions *during* the analysis, a challenge here is that the linearized data would constantly change. The paper on tailorable sampling conceptually explores mechanisms for progressively updating the linearization step, which could facilitate such an approach. Combining the two approaches, thus, could provide valuable means of automatically tailoring the sampling to the user interest in PVA.

4.2 REFLECTING ON THE RESEARCH QUESTIONS

This section sets the contributions in this thesis in context with the research questions posed in the Introduction in Chapter 1.

RQ1: *How can the sampling in PVA be tailored to bring out the desired information in partial visualizations?*

In response to RQ1, this thesis contributed tailorable sampling for PVA, which allows fitting the data partitioning step to analysis scenarios to improve the expressiveness of partial visualizations. This is achieved by conceptually modularizing the PVA sampling process along a three-step pipeline. Each step of this pipeline can be customized by analysts to adjust the sampling to their analysis scenario. Tailorable sampling improves the expressiveness of partial visualizations in PVA, as it allows analysts to adjust the order in which the dataset is processed, thus allowing to bring out those data that make the visualization as useful as possible as early as possible. The utility of this approach was demonstrated in a series of examples.

Furthermore, this thesis contributed a characterization of PVA sampling, which distinguishes it from other sampling approaches, in particular sampling for non-progressive VA. In non-progressive VA, sampling is a one-time operation that is performed once before running the analytic computation on the resulting sample, whereas in PVA, the sampling is a process that runs in parallel to the interactive visual analysis process. These two processes influence each other. On one hand, the sampling is tailored to the analysis process, and on the other hand, the sampling influences the order of the data shown in the partial visualization. This also means that PVA sampling is not static but an inherently dynamic process, which must be tailorable to changes in the analysis scenario. As demonstrated in the work on tailorable sampling, the modular structure of the sampling pipeline accounts for this, in that it not only allows tailoring the sampling *once*, but *dynamically* throughout the progression.

RQ2: *How can DOI functions in PVA be continuously updated to allow extracting the information needed for the task from partial visualizations?*

In response to RQ2, this thesis contributed strategies for enabling DOI functions in PVA, which help improve the effectiveness of partial visualizations. This is done by increasing the scope of interest values beyond the chunk they were computed in. On one hand, this entails increasing the scope for the computation of the next chunk to also account for already computed values, and on the other hand increasing the scope of already computed values to the data in the next chunk. Rather than requiring an exhaustive recomputation of interest values each time new data arrives, the proposed strategies allow reducing the computational overhead by (1) selecting a representative subset of data items for the computation of the next chunk and by (2) updating the interest values only for data items that are outdated. As a result of this optimization, those DOI functions that could previously not be applied in the progressive use case due to long computation times can now be used to make partial visualizations more effective in PVA.

RQ3: *How can steering be made more widely applicable across PVA scenarios to reduce the mental and computational efforts of using partial visualizations?*

In response to RQ3, this thesis contributed steering-by-example, a steering mechanism that allows focusing computational resources to complete partial visualizations for a selected data subspace of interest. Compared to existing steering mechanisms, steering-by-example works independently of the used visualization technique and does not require additional index structures to function. Steering-by-example only requires a set of data items as input, which can be easily generated for most visualization techniques. A decision tree classifier is then trained on this data to derive decision rules for distinguishing those items from the rest of the data. Adding these rules to the query that retrieves the next chunk then steers the progression towards similar data. Keeping the "entry barrier" low means that steering-by-example helps improve the efficiency of partial visualizations in PVA that previously did not fit the demands of existing steering mechanisms.

4.3 FUTURE RESEARCH DIRECTIONS

There remain many open research questions regarding making partial visualizations in PVA more useful, some particularly interesting ones are outlined below.

4.3.1 *Providing integrated library support to developers*

One main challenge for adapting useful partial visualizations — and PVA approaches in general — in the real world is efficiency, which is impeded by its large "entry costs". A working group at a Dagstuhl seminar on Progressive Data Analysis and Visualization back in 2018 already identified increase in conceptual and algorithmic costs as a clear "threat" to PVA [39, pp.36], in that the costs may outweigh the benefits of the progressive approach. The authors specifically note the increase in the design and implementation of progressive systems, and the additional overhead in terms of computation and user effort as potential fail criteria. Since then, new PVA approaches have been proposed, but implementation costs remain large, as most implementations require ad-hoc solutions. Therefore, implementing dedicated libraries for PVA and progressive analytic methods in general remains an important field for future work. Libraries like ProgressiVis [37], PV [68], and ProgressiveDB [13] are promising first steps in that direction, but they remain research projects and are mostly used experimentally.

One challenge will be integration, both *internally* to make already existing PVA approaches useable in concert, but also *externally* in that libraries for PVA should integrate with as little adjustment as necessary into existing (non-progressive) VA workflows. Moreover, Fekete et al. discussed open challenges for bringing PVA into the "real world" in a recent workshop paper [38] highlighting compressed bitmaps, data sketching, and online algorithms as promising technologies to build onto. PVA's future usefulness from a practical standpoint strongly depends on development efforts to lower the entry costs for scalable analytics through progressive means.

4.3.2 *Guidance for improving the usefulness in partial visualizations*

Another future challenge is providing conceptual support for the design of useful partial visualizations. The contributions in this thesis provide more control over the analysis process, but with more control comes more complexity: Analysts need to carefully consider how to tailor each step of the sampling pipeline, with which DOI strategies they want to identify interesting items, and supported by which steering mechanism they want to perform their analysis to end up with useful partial visualizations. In addition to the existing PVA approaches, this requires strong background knowledge from analysts, while also baring the potential for misconfigurations.

Complexity is inherent to many VA approaches, and the scientific literature has developed the concept of guidance in response. Guidance aims to actively resolve analysts' knowledge gap during interactive VA sessions [20]. Thus, guidance not only refers to leading analysts to the data they are interested in, but also refers to supporting the construction of the VA process itself.

A future research direction for improving the usefulness of partial visualization is to develop guidance to support analysts at all stages of the PVA process.

4.3.3 *Increasing the efficiency with progressive parameter space visualization*

Another open challenge in VA is that the parameters used in analytic computation influence the usefulness of the visualization. Even when running the computation progressively, it can take many attempts to find suitable parameters that fit the analysis scenario, as "getting it right" is non-trivial. One approach for reducing the number of restarts are parameter space visualizations, which provide a visual overview of the effect of potential parameter combinations on the computation result. This enables analysts to identify the suitable parameter combinations *before* starting their interactive visual analysis. However, computing many parameter combinations for large datasets can take a long time, making parameter space visualization itself a clear candidate for PVA.

A step in this direction was done in work adjacent to the thesis contributions together with Loeschcke et al. [74]. Therein, a matrix visualization encodes the utility of parameter configuration through a scenario-specific quality metric. In particular, the paper explores progressive parameter space visualization for time series data using Symbolic Aggregate approXimation (SAX) along two input parameters. The proposed approach also allows steering the progressive visualization, such that parameter combinations of interest can be prioritized. In the presented use cases (analyzing weather data and light curves), the visualization allowed identifying useful parameter combinations early on, thus making the parameter space visualization more efficient.

While these first results are promising, more work is necessary in that space. For example, one question is how to adapt the approach to different and more complex parameter spaces than the two dimensions discussed in that paper. Another question is how to adapt the approach to different data types and computations than those explored in that paper. The hope is that by addressing these questions will allow for more efficient — and, thus, useful — partial visualizations in PVA.

Part II

PAPERS

5 A PIPELINE FOR TAILORED SAMPLING FOR PROGRESSIVE VISUAL ANALYTICS

Marius Hogräfer, Aarhus University, Denmark
Jakob Burkhardt, Aarhus University, Denmark
Hans-Jörg Schulz, Aarhus University, Denmark

Proceedings of the International EuroVis Workshop on Visual Analytics (EuroVA), 2022, pp.49-53 *doi:10.2312/eurova.20221079.*

ABSTRACT

Progressive Visual Analytics enables analysts to interactively work with partial results from long-running computations early on instead of forcing them to wait. For very large datasets, the first step is to divide that input data into smaller chunks using sampling, which are then passed down the progressive analysis pipeline all the way to their progressive visualization in the end. The quality of the partial results produced by the progression heavily depends on the quality of these chunks, that is, chunks need to be representative of the dataset. Whether or not a sampling approach produces representative chunks does however depend on the particular analysis scenario. This stands in contrast to the common use of random sampling as a "one-size-fits-most" approach in PVA. In this paper, we propose a sampling pipeline and its open source implementation which can be used to tailor the used sampling method for an analysis scenario at hand.

5.1 INTRODUCTION

A common challenge for interactive visual analysis is bringing the user into the loop during long-running computations [34, 82]. A promising solution to this challenge is Progressive Visual Analytics (PVA) [39, 107]. Therein, analysts are presented with intermediate, incomplete results from these long-running computations, allowing them to gather early insights rather than having to wait until all data is fully processed. It has been shown that analysts in progressive systems often outperform those of non-progressive systems in terms of efficiency [127].

One common way of generating partial results is by chunking up the data into smaller pieces, and then incrementally computing and visualizing results for these pieces over time. In practice, this seemingly simple process turns out to be rather complex, since splitting

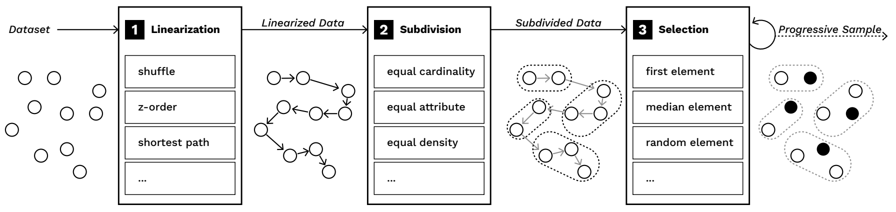


Figure 11: Our sampling pipeline for Progressive Visual Analytics, including some exemplar operators used at each step: (1) The input data is first normalized into linear order in the *linearization* step, (2) then that data is structured into groups with the *subdivision* step, and lastly, (3) elements from these groups are then picked in the *selection* step to form a chunk that is representative in the context of the analysis scenario.

up the data means that the interactive visual analysis process – inherently an uncertain and often exploratory endeavor – becomes even more uncertain: any patterns that analysts find in the visualization could change in the future, once more data is processed.

The general goal for sampling in PVA is thus to produce representative samples of the data that reduce the likelihood of the visualization changing over time. Yet, how exactly this "representativeness" is characterized depends on the analysis scenario. Prior work has thus proposed dedicated sampling methods that produce representative samples in particular scenarios. For example, Chen et al. present a method that preserves the local density and outliers in the visualization, as well as temporal coherence of subsequent chunks when progressively sampling for scatter plots [23]. Others have explored sampling methods for spatio-temporal data [115] or for prioritizing salient features of the visualization in the sampling [97]. Nevertheless, the default sampling method for scenarios, for which no such dedicated approach exists, remains random sampling. There are however some downsides to drawing random samples: (1) it can produce misleading visual artifacts on some visualizations [129], (2) it is a poor fit for certain tasks such as outlier analysis [23], where the data users look for are unlikely to be sampled early, and (3) it can fail to represent class distributions of imbalanced datasets.

We thus propose a sampling pipeline for PVA that can be configured to better fit the particular analysis scenario than random uniform sampling, providing a fallback option for analysis scenarios not yet covered by dedicated solutions. We introduce ProSample, a comparison tool for pipeline configurations, which allows comparing how the progressive results differ between two pipelines. Using ProSample, we showcase our sampling pipeline for three scenarios.

5.2 RELATED WORK

Dividing a dataset into chunks is an integral part to the PVA process and "getting it right" is important, since any flaws introduced by this chunking will be present at any downstream operation. For instance, Procopio et al. note the role sampling plays in reducing the error bars in the visualization, i.e., the uncertainty of the analysis results that analysts work on [94]. Given this importance, it is surprising that the most commonly-used sampling method remains random sampling, or shuffling the input dataset once and then chunking it in order, in cases where the data size causes random sampling to take too long.

Some prior work in the field of PVA has looked at specific sampling scenarios. Chen et al. present a progressive sampling method that maintains outliers, avoids overplotting, and ensures that consecutive chunks are coherent in progressive scatter plots [23]. Turkay et al. propose an adaptive sampling for progressive visualization of high-dimensional data, which dynamically adjusts the size of the (in their case random) sample to ensure that the computation produces new results within a certain time interval [110]. Another example is the selective wander join method proposed by Procopio et al., which addresses the challenges of sampling for complex database queries that contain data joins, i.e., the data is retrieved from multiple tables at the same time [95]. The method also lends itself for progressively sampling from groups of skewed data, which can be desirable for prioritizing data of interest.

Under the term "approximate query processing" (AQP) and "online sampling", dedicated sampling methods have been developed to better capture data qualities like skew or error bounds. Sample+Seek [30] and BlinkDB [2] are examples of sampling methods for reducing or bounding errors and response times of queries on large datasets. An example for applying AQP in PVA is so-called optimistic visualization [80], which helps analysts recover from false conclusions drawn from the approximate data. This is done by visualizing the difference between the approximate result at the time analysts drew their conclusions and the current state of the progressive computation, allowing analysts to identify any discrepancies. Others use features of the visualization to optimize the sampling. Rahman et al. present a sampling method that aims to retrieve data for salient features in the visualization first, then sampling less salient features [97]. Park et al. present visualization-aware sampling, which uses prior knowledge about the visual encoding (scatter plots or maps) to produce an appropriate sample from large datasets, suggesting that running their algorithm multiple times may be beneficial to incremental visualization [89]. Wang et al. present STULL, a progressive sampling method for spatio-temporal data that retrieves samples that show the same distributions as in the full dataset [115].

5.3 A SAMPLING PIPELINE FOR PVA

On one hand, the related work shows how important it can be to have specific sampling techniques tailored to a particular scenario at hand. On the other hand, it also shows that we are far from having such a specific sampling technique for all possible scenarios. This is why we introduce a sampling pipeline for PVA, which is shown in Figure 11. The main design goal of this pipeline is flexibility – i.e., to be adaptable or configurable to fit a wide range of possible sampling scenarios. Yet at the same time, we also want to be able to reuse parts of a sampling technique across different scenarios. Hence, we strive for a standardized process where we make as little assumptions about the analysis scenario as possible apart from the format of the inputs and outputs of each step in the pipeline. The pipeline steps can then be concretized using different operators that are reusable across different scenarios [49].

Our proposed sampling pipeline for PVA consists of three steps: linearization (which normalizes the input data), subdivision (which structures the linearized data), and selection (which generates the chunks from that structure). We detail these three steps below and provide general considerations for configuring each step.

5.3.1 *Linearization*

The linearization step normalizes the input data into a simple list of elements. This is necessary, as the type of the dataset influences the way it can be processed (e.g., graph data requires different algorithms than tabular data or geospatial data). By normalizing the input data, we essentially remove these specific characteristics from the input dataset, reducing the assumptions we have to make about the data at later stages in the pipeline. This increases both the flexibility and reusability of downstream operators.

The proposed linear structure is conceptually simple and most data types can be transformed into it: a graph can be linearized along its shortest path, geospatial data can be linearized using a space-filling curve like the z-order curve, hierarchies can be linearized using a BFS or DFS traversal, and n-dimensional data can be linearized using a knn-based heuristic to solving a traveling salesman problem over the data. The concrete operators used in this step are thus very much dependent on the type of the input data.

Once in linear form, it is further possible to reorder the data to account for requirements of the analysis scenario. For example, we can randomly shuffle the list to overcome sorting biases, or purposefully sort the list by some attribute value. Having the data in a certain order can be beneficial for operators in later steps of the pipeline, yet it comes at the cost of additional computation time.

5.3.2 *Subdivision*

The subdivision step takes the linearized data and partitions it. This can be seen as cutting-up the long list of all data into a set of smaller consecutive lists of data items.

This subdivision is very much dependent on the task to be carried out. If the task is still unspecified, we can simply subdivide the data into sets of equal cardinality – i.e., the same number of items in them. For an overview task that is to first show the extent of the data and not so much its density, we can subdivide the data into sets that maximize coverage over a given value range. Depending on which attribute's value range is used, this puts additional focus on a data dimension of interest. When exploring spatially clustered data items, we can run Lloyd's algorithm on the linearized data [73], essentially computing a 1D k-means clustering on it.

The number of groups we subdivide the data into is scenario-specific. For example, if analysts want to just get a rough overview of the data, we can set the cardinality of the groups so that the progression produces results within acceptable response times. When using a nominal dimension to facet the data into groups, the number of facets is an appropriate fit. When considering multiple dimensions, we might use the number of classes found by the clustering algorithm as number of groups.

A hierarchical subdivision (e.g., first dividing by coverage for one data attribute and then subdividing each set further by another data attribute) is possible.

5.3.3 *Selection*

The selection step then defines a strategy for constructing chunks from the structured data that most benefit the analysis scenario. To that end, it subsequently selects items from the subdivisions that best match the user interest in the data.

The choice of a selection operator depends in many ways on the user role as it is defined by Micallef et al. [78]. A *progressive observer* who monitors an evolving progressive visualization is interested in seeing a reasonable representative of the full dataset at any time. In this case, the selection could thus simply draw the medians from each subdivision. This is different for a *progressive searcher*, who uses the progression to quickly find an answer without having to look at all the data. If the searcher is interested in extreme values, the selection should draw min/max values from each subdivision. If the searcher is interested in the largest clusters, the selection operator should draw exclusively from the largest subdivision generated by the k-means operator mentioned above and then work its way downward to the smaller clusters. Finally, the *progressive explorer* uses PVA to be able to quickly switch between different configurations at runtime, depending on observations and insights gained from the partial results. Hence, it is not a single selection strategy in which the explorer is interested, but in the ability to switch between different selections and their parametrizations to adjust the incoming data chunks to their current needs. For example, the size of the selection (and thus of the resulting chunks) may need to be changed to meet a desired response time.

5.4 USAGE EXAMPLE OF THE SAMPLING PIPELINE

We showcase the versatility of our approach by first introducing its implementation *ProSample* and then applying it to a use case for which we configure and compare three sampling pipelines.

5.4.1 *ProSample*

We make available an open source implementation of our pipeline, called *ProSample*, which allows analyzing the effect of different pipeline configurations on the PVA process. In ProSample, the user can configure two sampling pipelines, which are then used to simultaneously process a dataset, showing the results in side-by-side views as regular or binned scatter plots. An optional third view encodes the delta between the two views in a binned scatter plot. The views can be explored with zoom and pan, using linked navigation. The interface of ProSample is implemented using D^3 [14] and runs in current browsers. Our implementation of the backend providing the configurable pipelines is done in Python, using the numpy package [111]. The code for ProSample is publicly available at `https://github.com/vis-au/prosample`

5.4.2 *Scenarios*

The scenarios are based on a dataset of mountain peaks from OpenStreetMap [88], which contains about $650,000$ items, providing longitude, latitude, and number of edits. For comparability, we sample one element from each group of an `equal cardinality` subdivision into $10,000$ items, and the selection operations consider only the dimension containing the number of edits. We precomputed linearizations.

We begin by configuring the sampling for an **overview** task. We use the `random` linearization to normalize the spatial dataset and structure it into groups using the `equal cardinality` subdivision, since we are still unfamiliar with the underlying dataset. As selection method we also set the `random` strategy, so that items per chunk are randomly picked from each subdivision group. We can see how the progressive visualization in ProSample early on shows the outlines of the conti-

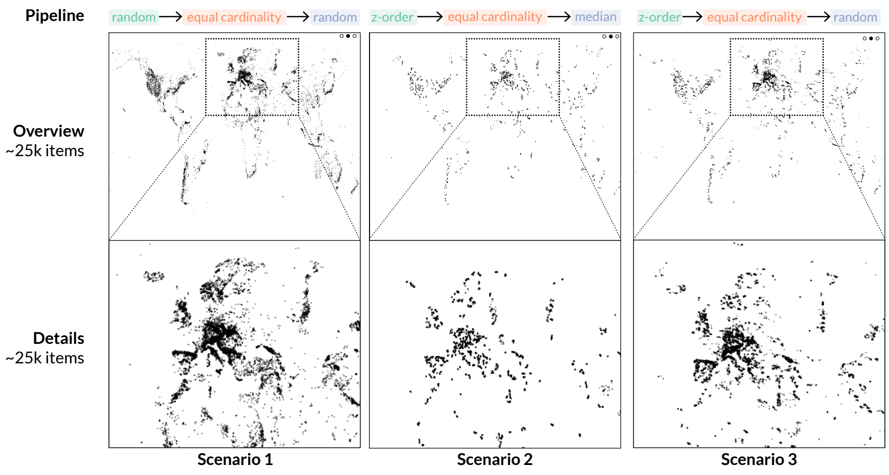

Pipeline
random → equal cardinality → random
z-order → equal cardinality → median
z-order → equal cardinality → random
Overview
~25k items
Details
~25k items
Scenario 1
Scenario 2
Scenario 3

Figure 12: The three scenarios discussed in the paper, showing the used pipeline configuration in terms of linearization, subdivision, and selection strategies, as well as an overview and detail view of the mountain peaks dataset after processing around 25 thousand items.

nents, and we can also quickly identify regions on the map that contain many mountain peaks, such as Central Europe, the Himalayas, and the Andes (see highlighted regions in Figure 12). If we were unfamiliar with the dataset, the first observation lets us quickly notice that the dataset contains spatial data, collected on a global scale, and the second observation allows us to identify that the measured points are mountain peaks.

Based on these insights, we move on to the next scenario, were we tailor the pipeline for a **density** analysis, in which we look first and foremost for regions with many mountain peaks. We do so by using a `z-order` linearization, which maintains spatial proximity between items in the normalized order, meaning that subsequent points in that list not are indeed located close to each other. Thus, when using the `median` selection operator, we will sample dense regions of the data.

We notice that the data in early chunks is “clumped” into highly dense regions – which is exactly what we wanted, but it misses the contextual information of the sparser regions. To also yield this **context**, we adjust the configuration again by exchanging the selection step for the `random` operator, which selects items across the groups defined by the subdivision. We can see the effect in the zoomed-in views in Figure 12, in that the sampled items are less “clumped” as before. With this context, we can for example identify the Alps in the context of Central Europe.

5.5 CONCLUSION AND FUTURE WORK

We have presented a sampling pipeline for PVA, which can be used to tailor the sampling process to the scenario at hand. We demonstrated the flexibility of this pipeline in three scenarios, through the comparison tool ProSample that is available as open source.

Having a framework for generalizable PVA sampling in place opens up space for future applications. For instance, we want to explore guidelines for choosing the most beneficial sampling strategy for a particular PVA scenario, that go beyond general considerations as we outlined in Section 5.3. This will require more qualitative, and certainly more quantitative evaluation of our framework. Qualitative evaluations could widen the scope to also consider the dedicated progressive sampling techniques presented mostly from database perspectives (see Section 5.2), while quantitative evaluations could measure the performance of progressive sampling using our framework, compared to these existing techniques.

Future work also needs to improve the practical implementation of our framework, extending the operators that are available so far in ProSample. Tool support is a general challenge in PVA, with only few research-focused frameworks like ProgressiVis [37] and P5 [68] in existence, and therefore most research being conducted on custom implementations. A progressive sampling library could however be a starting point towards more reusable solutions.

Lastly, we also want to expand on the principal considerations of sampling in PVA. In this paper, we have discussed sampling solely as a method for dividing the input dataset into smaller chunks that are then passed downstream to a PVA pipeline. However, given the thread-based process model presented by Schulz et al. [102], dividing the data into chunks could also happen at different points in the PVA pipeline. In fact, their model suggests that each operator in the progressive pipeline can freely decide on when to process its input. This raises research questions regarding appropriate sampling pipelines dedicated for the view rendering step, best practices for combining operators using different sampling pipelines, or even having more than one pipeline per operator. Our framework provides parts of the groundwork towards these questions.

ACKNOWLEDGEMENTS

This work has been funded in part by the Innovation Fund Denmark through the Grand Solution project *Hospital@Night*.

6 TAILORABLE SAMPLING FOR PROGRESSIVE VISUAL ANALYTICS

Marius Hogräfer, Aarhus University, Denmark
Hans-Jörg Schulz, Aarhus University, Denmark

Submitted to IEEE Transactions in Visualization and Computer Graphics.

ABSTRACT

Progressive Visual Analytics (PVA) allows analysts to maintain their flow during otherwise long-running computations, by producing early, incomplete results that refine over time, for example, by running the computation over smaller partitions of the data. Generally, these partitions are created using sampling. The goal for sampling in PVA is, therefore, to draw samples of the dataset such that the progressive visualization becomes as useful as possible as soon as possible. What makes the visualization useful depends on the analysis task and, accordingly, some task-specific sampling methods have been proposed for PVA to address this need. However, as analysts see more and more of their data during the progression, the task in PVA often changes, which means that analysts need to restart the computation to switch the sampling method, causing them to lose their analysis flow. This poses a clear limitation to the proposed benefits of PVA. To this end, in this paper we propose a pipeline for PVA-sampling, which allows tailoring the data partitioning to analysis scenarios by changing out modules, and in a way that does not require restarting the analysis. We for the first time characterize the problem of PVA-sampling, and also formalize the pipeline in terms of data formats, discuss on-the-fly tailoring, and present additional examples demonstrating its usefulness.

6.1 INTRODUCTION

Visual Analytics (VA) aims to combine the computational power of modern hardware with the reasoning skills of human analysts for data analysis through visualization, with analysts configuring the analytic computation based on observations of the resulting visualization. To be effective, VA requires that updates after an interaction appear within interactive response rates of around 1 s, as analysts otherwise lose their analysis "flow" [33]. One challenge to interactivity in VA is the increasing size of datasets, which slows down computation

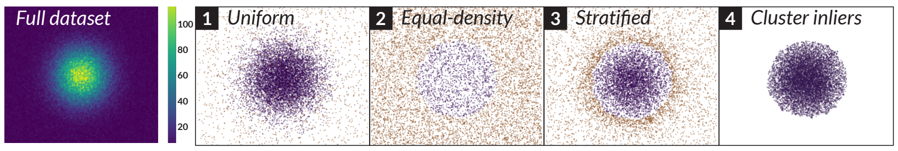


Figure 13: Depicted is a toy dataset containing 100k items with two numeric attributes of normal distribution (encoded along the x and y axes) and a Boolean attribute. On the left, the distribution of the x and y attributes is depicted in a binned scatterplot. Four samples ($||\mathtt{sample}|| = 10\text{k}$) are drawn from this dataset and depicted in subfigures (1) to (4), each of which is tailored to fit a specific analysis task. The Boolean attribute is encoded as the color. Each sample brings out different aspects of the dataset, showing the importance for tailored sampling in PVA.

times, thus making VA ineffective. To nevertheless bring the benefits of the interactive visual analysis to large datasets, one approach is to split the dataset into smaller partitions and to then run the analysis on those smaller partitions, showing partial results to analysts. This so-called Progressive Visual Analytics (PVA) approach [107], puts analysts "back into the loop" of long-running computations, allowing them to regain the flow. Benefits of this approach have, among others, been highlighted by Zgraggen et al. [127] who show users of progressive systems to clearly outperform those using traditional "blocking" systems. Beyond early insights, additional benefits of PVA include the interactive parametrization of long-running computations setting parameters on the fly, the ability to steer computations towards data subspaces of interest, early termination (stopping a long-running computation early on), and the ability to observe how otherwise intransparent "black box" computations evolve (see the detailed review by Angelini et al. [6]).

Nonetheless, for PVA to be beneficial, the partial results shown to users need to reflect the final result, and any patterns in these partial visualizations should also remain stable throughout the progression. In turn, the goal for partitioning the data in PVA (which we refer to as PVA-sampling) is to *make the visualization as useful as possible as early as possible*. However, we identify two drawbacks in the current approach to partition the data, which limit the effectiveness of PVA.

First, the notion of usefulness in the sampling goal stated above highlights a relation to the analysis task: What data should be in a "useful" sample depends on what analysts are going to do with it. For example, we can consider the multivariate toy dataset depicted in Figure 13, containing two normally distributed numeric attributes (encoded as x and y positions) and a Boolean attribute (which is encoded as color). Depending on what task an analyst wants to perform on that data, there are different ways for how to make the sampling of that data most useful: For example, to gain an initial overview of the data, it makes sense to draw a uniform sample that helps depict the distribution of all three attributes. In the sample depicted in subfigure (1), the densely populated region in the center of the plot stands out. On the other hand, to analyze the local distribution of the Boolean attribute, it is more useful to sample the data along a regular grid, such that the density in each grid cell is even throughout the sample, which puts the focus on the Boolean attribute. In subfigure (2), we clearly notice the circular border between the two attribute classes. Another task could involve training a classifier on the Boolean attribute. Then, a stratified sample is more useful, such that both attribute values are evenly represented in the training data, as depicted in subfigure (3). Lastly, analysts may also focus their analysis exclusively on one facet of the Boolean attribute, prioritizing these items in the sample as in subfigure (4). The challenge in all this is that the current state of the art sampling mechanism in PVA remains random sampling, but, as we saw above, there are many cases where this approach impedes the efficiency of PVA, as analysts would need to wait before the results computed over random samples become useful to them. While approaches have been proposed to provide scenario-specific sampling, these generally do not translate well beyond the scope they were designed for, so more tailorable solutions are needed.

In addition to the lack of tailorability and in contrast to sampling for regular VA tasks, analysts in PVA can dynamically change the course of their analysis mid-computation, accounting for insights gathered from the partial visualizations rather than restarting the analysis. This means that both the data type *and* the task may change. For instance, an analyst may begin by passively observing the progression over the multivariate dataset from our previous example to get a spatial overview of their dataset, and then move on to analyze "inlier" items along the Boolean attribute, once they identified that region as interesting. In non-progressive VA, making this switch is not an issue, as analysts can reconfigure and rerun the entire analysis whenever their task changes. However, in PVA, the analysis is an ongoing process and, thus, configurations need to take place on-the-fly. This is a challenge for the sampling, as the two tasks in the example above require completely different data: For supporting the overview task, samples should evenly represent geospace to bring out general patterns in the data, while in the outlier task, the sampling needs to bring out rare items along a numeric attribute. This shows that the traditional "fire-and-forget" approach to sampling, where the sampling is configured *once* before starting the analysis, no longer applies in PVA, exactly *because* analysts can now interact with that analysis. Yet, existing sampling techniques are generally tailored to specific tasks on specific data types. Thus, when the task or the data type of interest changes like in the example above, analysts need to restart to change to a dedicated sampling technique, breaking exactly that flow that the progressive analysis was supposed to ensure in the first place. Thus, a new approach is needed that fits the dynamic demands of PVA.

To this end, we identify two main challenges for the effectiveness of PVA: (1) Current sampling algorithms cannot be tailored to the task, requiring dedicated implementations whenever the standard method of random sampling falls short, and (2) the sampling cannot be adjusted while the computation is ongoing, whenever analysts change their task based on new insights, which reduces the effectiveness of PVA. In this paper, we address these challenges by introducing a new approach to PVA-sampling, which modularizes the sampling process and thereby allows tailoring it to the requirements of tasks and dataset in a way such that this tailoring can also take place without stopping the computation.

This paper expands on our 2022 EuroVA workshop paper [52], in which we first proposed the idea of a sampling pipeline for PVA. Concretely, this extension consists of the following main contributions:

- We added a characterization of PVA-sampling along its unique challenges that clearly distinguishes it from regular sampling.
- We added a formalization of the pipeline by defining its steps as transformations between data formats along the operator design pattern for visualization.
- We added dynamic tailorability as an additional requirement for PVA-sampling and show how the pipeline can be used to this end.
- We added a series of new examples on a real-world dataset, showing how the pipeline enables tailored sampling, but also how it can be used to recreate existing approaches.

The remainder of this paper is structured as follows. First, we characterize PVA-sampling by outlining its unique requirements and distinguish it from related sampling approaches. Then, we introduce the modular sampling pipeline, using a running example to demonstrate the effect that each step has on the final sample. Afterwards, we show how the pipeline allows for on-the-fly tailoring of the sampling, and discuss additional benefits and limitations.

6.2 CHARACTERIZING PVA SAMPLING

PVA supplies analysts with early, partial visualizations to bring the benefits of an interactive visual analysis to long-running computations. One approach for creating these partial visualizations is to partition the data into chunks using sampling. Sampling for PVA (which we refer to as *PVA-sampling* in this paper) is, however, quite different from regular sampling. This is because in PVA, sampling is an ongoing process that runs in parallel to the visual-interactive analysis, while regular, non-progressive sampling is an operation carried out once. As a result, regular sampling determines *if* an item appears in the sample and thus will become part of the analysis – a binary selection –, whereas PVA-sampling determines *when* a data item becomes part of the analysis – a ranking of data that prioritizes data items to be pushed through the analysis pipeline before others (see Figure 14). This seemingly simple conceptual difference yields three unique requirements for a useful PVA-sample:

The first requirement is the frequency at which the process runs and draws new samples. PVA-sampling needs to continuously produce samples throughout the analysis, such that new updates to the visualization arrive within the so-called human latency limits. Angelini et al. specify these limits depending on the task at three levels, with under 1s for perceptual updates, 1s for immediate responses, and up to 10s for task completion [6]. Achieving these update rates is one of the main motivations for using PVA, as it allows analysts to maintain their analytic flow even during long-running computations [33]. This stands in contrast to regular sampling, where a single sample is drawn before the analysis, meaning that the human latency limits are not a consideration for the sampling. A side effect of this requirement is that PVA-sampling sometimes produces samples that are too small to be statistically representative. It is *because* the sampling is a process that these samples nevertheless become useful to analysts eventually, once enough data has been sampled.

The second requirement is that priorities are given to different parts of the data as to what to sample first and last. PVA-sampling defines for all items, *when* they are selected, and so all data is eventually part of the analysis – unless, of course, analysts terminate the computation beforehand. Regular sampling for VA, on the other hand, does

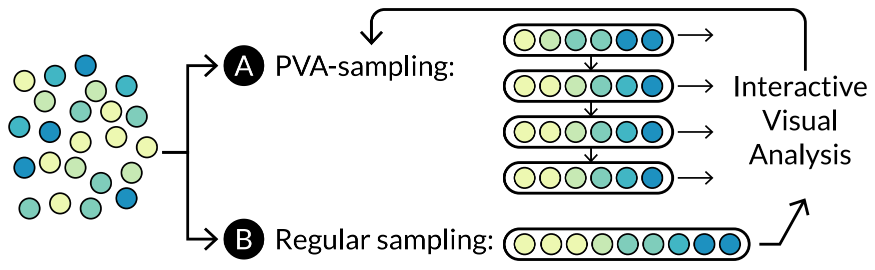


Figure 14: Demonstrating the difference between PVA-sampling and regular sampling: Regular sampling (B) and the analysis of the data sample happen consecutively: The sample is drawn, and then analyzed. Thus, regular sampling defines for each item, whether it is part of the analysis or not. In contrast, PVA-sampling (A) is a continuous process taking place in parallel to – and potentially also influenced by – the analysis. It draws new samples (so-called chunks) from the full dataset up until the user stops this process or all data has been sampled. Thus, PVA-sampling defines for each item, when it becomes available to be shown and analyzed.

not prioritize the data, but instead it defines for all items *if* they are selected as part of the sample. All items that are not part of the sample are, therefore, also not part of the analysis. The way in which PVA-sampling prioritizes certain data items depends on what makes the visualization useful as soon as possible, meaning that it depends on the analysis task (as outlined in the Introduction).

The third requirement for PVA-sampling is the flexibility of adjusting the sampling process while it runs, in order to sync it with the visual analytic process that is concurrently being carried out: as one process changes, so must the other. One side of this dependency is that sampling depends on the analysis, in that the task characterizes what items are useful. But it also means that the analysis is influenced by the sampling, as analysts gain new insights from the partial results in the samples, which again influences their task. For PVA-sampling to be flexible, this means that the process can be dynamically adjusted (i.e., without restarting) to tailor the data prioritization to changing tasks. This is not a requirement for regular sampling in VA: The sampling terminates before the analysis. Thus, in order to tailor it to the task, the sampling step must be rerun to create a more fitting sample.

6.3 RELATED WORK

In this section, we compare PVA-sampling with related sampling methods in the literature.

6.3.1 *Sampling as an operation*

In Section 6.2, we distinguished PVA-sampling from regular sampling for PVA, in that regular sampling is an operation that concludes, before the interactive visual analysis is run. Regular sampling for VA is commonly used to reduce the complexity of large datasets, in cases where approximate results are as useful as seeing the full picture [67]. This can be desirable for many reasons, including reduction of computation time for complex analyses [67], and also clutter reduction in view space [31]. In contrast to PVA-sampling, only one "final" sample is used to conduct the analysis. Therefore, it is generally important that this sample is statistically representative of the dataset, so that the insights gathered from it also apply to the rest of the data. Olken and Rotem provided an early survey of sampling methods in 1990 [86], and since then, many more sampling algorithms have been proposed. To the best of our knowledge, what they have in common, though, is that tailorability beyond a particular analysis scenario is generally not considered. This makes "regular" sampling algorithms a poor fit for PVA, as switching the sampling method to tailor it to a new task requires a complete restart of the computation.

Another related method is active learning [104], where the goal is to find a "best" training data subset, for which a model performs best than when trained on all training data. Rather than selecting "clear-cut" items far away from the decision boundary that clearly belong to a class, active learning aims to sample borderline items for which the prediction certainty is low. The goal is to sharpen the decision boundary around those edge cases, as those have a stronger impact on its performance. Active learning thus tries to make the model as accurate as possible as early as possible. Human factors including the latency limits are, however, generally not considered for the sampling.

6.3.2 *Sampling as a process*

There are other examples of sampling processes, where the analysis and the sampling are iterative, in that the sampling is adjusted based on insights gained from the sample, and vice-versa. One method similar – not only in name – to PVA-sampling is so-called progressive sampling, where an increasingly larger sample is drawn until a quality metric computed with that sample is reached or until a metric no longer improves. Usually, progressive sampling is used to reduce model training times, by determining the smallest necessary training sample from a large dataset, beyond which the prediction accuracy no longer (noticeably) improves [96]. Starting from an appropriate initial sample size [48], the model is fully retrained on a *progressively* larger sample. Progressive sampling shares some similarities with PVA-sampling, in that the goals it make the model as useful as possible as soon as possible. In contrast to PVA-sampling, though, progressive sampling generally does not consider the human latency limits, since it is an automated, metric-driven process that does not rely on user input after it is launched.

Another approach using sampling as a process is streaming sampling, where the goal is to run an analysis over a potentially infinite data stream. One of the major challenge here is, thus, maintaining the "ground truth" data the sample is drawn from. For example, the well-established reservoir sampling method [113] from 1985 produces a uniform sample over all data that has been processed so far, while only requiring to keep a sample in memory. A more recent example is the approach by Losing et al. [75], who use clustering to summarize the data stream to a set of representatives. Similar to PVA-sampling, we need to continuously sample, yet, analysts may not end up seeing *all* data, as only some new elements land in the sample. Another difference to PVA-sampling is that the data stream may be infinite and the result may not converge.

6.3.3 *PVA-Sampling approaches*

Prior work has investigated some aspects of PVA-sampling as characterized in Section 6.2. The standard sampling approach in PVA literature is arguably random sampling without replacement [10, 59, 68, 110, 127], as PVA is often proposed as an interactive method for the overview task. Tailored approaches have also been proposed. Most recently, Chen et al. presented sampling for progressive scatterplots [23], using three criteria to define a useful sample: preserve temporal coherence between successive samples, preserve the relative density and outliers, and achieve sufficient efficiency to retrieve samples within the latency constraints of PVA. Another approach is the work by Turkay et al., who introduced a method for adapting the size of the sample dynamically, such that the visualization is updated within a specified interval [110]. A common consideration for PVA-sampling is to reduce the error in the partial visualization. One example is the work by Rahman et al. who present an algorithm that prioritizes salient features in treemaps and line charts [97], or Sample+Seek [30] and BlinkDB [2], two sampling approaches that reduce and bound errors and response times of certain query types on large datasets. Another example is the selective wander join method proposed by Procopio et al. [95], which addresses the challenges of sampling for database queries containing data joins that also apply a filter on the data, achieving interactive sampling speeds in these cases.

The diversity of these techniques illustrates the benefit of (and need for) having tailored sampling algorithms, yet, it also shows that existing sampling techniques use dedicated, custom implementations as reusing parts of approaches to transfer them to other scenarios is generally not considered. Moreover, exchanging the sampling method mid-analysis is generally not discussed. This is why we propose a sampling pipeline for PVA, which modularizes the sampling process, allowing to tailor it to the requirements of an ongoing analysis, while increasing reusability of parts of the sampling process across scenarios.

6.4 A PIPELINE FOR TAILORABLE PVA-SAMPLING

Here, we introduce our tailorable sampling approach for PVA using a pipeline. We first derive the steps of the pipeline from input and output formats and then discuss each step in detail, demonstrating their impact on the sampling with a running example.

6.4.1 *Modularizing PVA-sampling along data formats*

A suitable conceptual foundation for our purposes is the Operator Pattern formulated by Heer and Agrawala, which enables “flexible

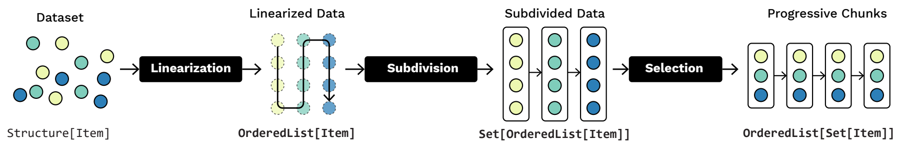


Figure 15: Enabling tailorable PVA-sampling using a pipeline that structures the sampling process into three steps (linearization, subdivision, and selection). The steps are depicted here along the data format they operate on: The linearization takes in the input data structure and transforms it into linear format, which is then subdivided into bins in the subdivision step. The last step then produces the chunks forwarded into the PVA process by progressively selecting appropriate items from each bin.

and reconfigurable" output [49]. We adapt their visualization-centric idea to sampling, in that each step of the pipeline is a module "that performs a specific processing action, updating the contents of the [sample] in accordance with a data state model" [49]. Each step of the pipeline, thus, applies a transformation on the input dataset analysts are working with, transforming it to a specific output format (i.e., the data state model in the above citation), finally leading to a series of so-called chunks. In relying on the operator pattern, we can make the complex PVA-sampling process tailorable to analysis tasks. It also allows us to represent transformations of different complexity, as well as intermediary operations that operate within one step (i.e., data state). Tailoring the sampling then means modifying these transformations used along the pipeline, and because transformations conform to the same input and output formats, we increase reusability between scenarios.

From a high-level perspective, PVA-sampling generally transforms the input dataset into ordered chunks. These chunks are then forwarded to the PVA process [68]. We can describe that input dataset as an arbitrary structure defined over a set of items $\mathtt{Structure}[\mathrm{Item}]$. We purposefully do not prescribe a particular data structure like table or graph here to keep the sampling pipeline independent of them, and we also keep the data type of *Item* arbitrary for the same purpose. On the other end of the process, the chunks produced by the sampling are a list of subsets of the input dataset $\mathtt{OrderedList}[\mathtt{Set}[\mathrm{Item}]]$. These chunks – that is, each $\mathtt{Set}[\mathrm{Item}]$ – are disjoint and for every item in the dataset, the sampling assigns a position in exactly one chunk. A PVA-sampling method is a function that transforms data from this input to that output format, and the sampling pipeline P must therefore conform to the following high-level format:

$$P : \mathtt{Structure}[\mathrm{Item}] \to \mathtt{OrderedList}[\mathtt{Set}[\mathrm{Item}]]$$

In order to make the complex, monolithic transformation P tailorable, we modularize it into three steps (*linearization*, *subdivision*, and *selection*). In the first step, the data is put into linear order, which is then

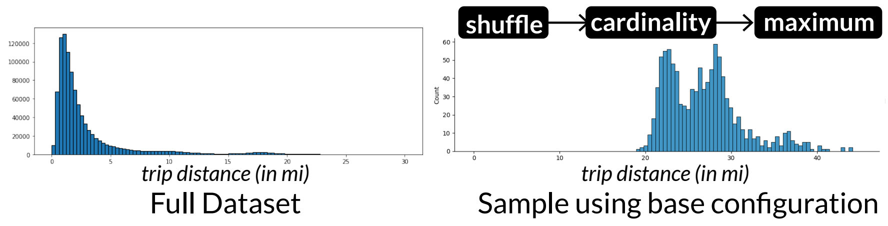


Figure 16: Left: Distribution of the *trip distance* attribute in the full dataset used in the running example, showing a clear spike in short distance taxi rides. Right: The distribution of the same attribute in a sample produced with the base pipeline ($||\mathtt{sample}|| = 10k$), showing a noticeable shift in the distribution towards longer trip distances.

subdivided into bins, from which the chunks are then assembled in the last step, selecting the most appropriate item from each bin. The data formats and steps are summarized in in Fig. 15.

RUNNING EXAMPLE: We introduce each step of the pipeline both conceptually and practically using a running example, demonstrating the effect of different operators at a particular step on the final sample. Concretely, we sample the 2018 Yellow Taxi trip dataset of taxi rides in New York City. This dataset contains around 112 Million items, each representing a taxi ride along numeric (e.g., trip distance), categorical (e.g., pickup and dropoff zone codes), and temporal (e.g., pickup and drop-off time) attributes. For illustrative purposes, we enriched this dataset with geospatial attributes for pickup and drop-off locations, by generating random locations in the polygons belonging to each zone code. Its size makes this dataset a clear candidate for a progressive analysis, and along its diverse attributes there are many interesting patterns to explore, thus requiring (dynamically) tailorable sampling. On this data, we use the following "base" pipeline as a running example: *random* linearization → *cardinality* subdivision → *maximum* selection. The linearization strategy puts the data in random order, the subdivision splits it up into bins of equal size, and the maximum strategy selects the greatest value along the *trip distance* attribute. This pipeline is configured to support an analyst interested in spatial distribution of extremely long taxi rides across the dataset, such that at every chunk it selects the maximum values of the *trip distance* attribute in the dataset. The ground truth distribution as well as the distribution yielded by the base pipeline is depicted in Figure 16. In the examples, we show how to further tailor the sampling to the task by making adjustments to this base setup at every step. Specifically, we demonstrate the impact on the *first* sample of 10k items using a particular pipeline.

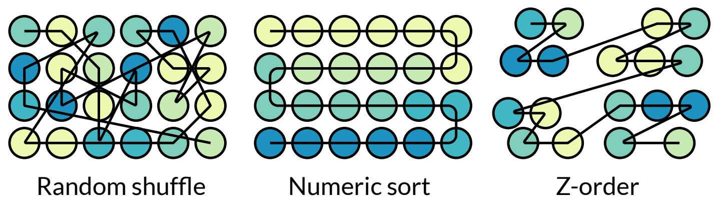


Figure 17: Three examples for linearizing the same dataset: Random shuffling, sorting by a numeric attribute, and sorting spatially along a z-order curve.

The code for the pipeline implementation we used, the data preprocessing we applied, as well as computational notebooks for reproducing the figures included in this section can be found on GitHub[1].

6.4.2 *Linearization: Putting the data in order*

In the first step, we harmonize and linearize the data into the standardized format `OrderedList[Item]`. This is necessary, as the type of the dataset influences the way it can be processed (i.e., a graph dataset needs to be treated differently than tabular data). Thus, by harmonizing the data into the linear list format at the first step, we allow the rest of the pipeline to be largely independent of the input data structure, while also increasing the reusability and flexibility of downstream operators.

How to appropriately linearize the dataset depends on the analysis scenario. The harmonization aspect is mostly influenced by the data structure. Because of the simplicity of the linear format, there often exist multiple linearization algorithms for a particular data structure, which means that as long as a fitting algorithm exist, we can harmonize many data types into the pipeline. Hierarchical data, for example, can be linearized using traversal strategies like depth-first search, geospatial data can be linearized using space-filling curves like the Hilbert [28, 131] or z-order curve [129, 133], and graphs can be linearized along their shortest path using a traveling salesman heuristic [87].

Once in linearized form, we can further reorder the items based on its attributes. Depicted in Figure 17 are three examples of strategies for sorting data based on their numeric values. The first strategy shown there is *random shuffling*, which puts the data in random order. Shuffling is widely applicable, as it does not use any item-driven metrics to sort the data, but instead makes as little assumptions about the data as possible. This makes shuffling also a good option whenever analysts' interest in the data is not clear, yet, or whenever analysts want to overcome unwanted sorting biases in the dataset. The second strategy depicted is *numeric sorting*, which as the name implies, or-

1 https://vis-au.github.io/prosample

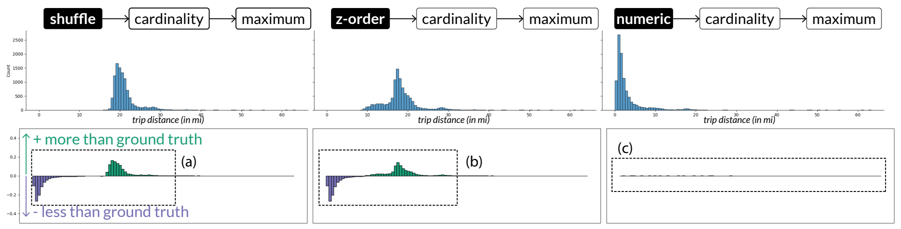


Figure 18: Impact of the linearization strategy on the sampling ($||\mathtt{sample}|| = 10k$). Depicted in the first row is the distribution along the trip distance attribute for *random shuffling*, *sorting with a z-order curve* along the pickup location attribute, and *sorting numerically* by the trip distance. The second row shows the relative difference in % compared to the distribution in the full dataset. We notice how the pipelines using shuffling and z-order linearizations produce samples that are clearly skewed towards larger values (a, b), while the numeric linearization maintains the original distribution (c).

ders the data by a numeric attribute of interest. Sorting means that similar items appear after one another in linearized format, which makes it easier to tailor the sampling to patterns found along that attribute, for example by sampling for similar values or by prioritizing outliers. For spatial data, we can use space-filling curves to put it in order, for example in *z-order* as depicted in the third example. Analyzing data based on spatial proximity is an important requirement in the analysis of spatial and geographic data, so space-filling curves allow tailoring the sampling accordingly.

EXAMPLE: Depicted in Figure 18 are the effects of these three linearization strategies on the base pipeline. In the first configuration, we want to make as little assumptions about the underlying data in the linearization which is useful for analysts unfamiliar with a dataset, thus we use *random* shuffling in the linearization step. Comparing the resulting sample to the ground truth by the trip distance distribution, we notice a clear skew towards larger values. This makes sense, as every item has the same probability for appearing in a subdivision bin, and therefore, when selecting maximum values from each bin, the sample will contain mostly larger values. To focus the analysis on the spatial distribution of the trip distance attribute, we can use the *z-order* linearization, resulting in items to appear after one another in the linearized data if their ride pickup locations are located close to each other. Thus, the subdivision bins created from this linearization are spatially ordered, and again, we see that the depicted sample distribution is skewed towards larger values, suggesting that longer rides are spatially spread out throughout the dataset. In our third configuration we *sort* the data by the attribute of interest, which means that the bins created by the subdivision contain items of similar values. The effect on the distribution is that the sample largely re-

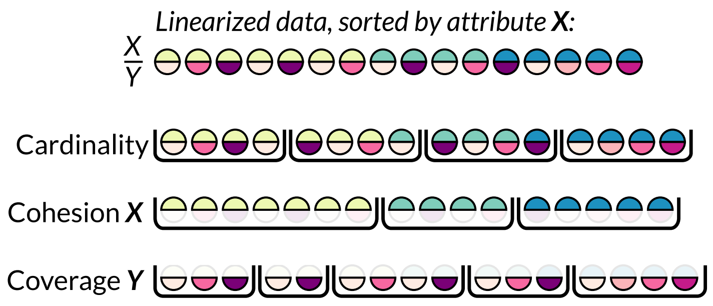


Figure 19: Examples for subdivision strategies over the same, two-dimensional linearized data: cardinality (splitting into equal-sized bins), cohesion (splitting at greatest differences in successive values, along the same attribute as used for the linearization), and coverage (splitting every time min and max values are found, along a different attribute than as used for the linearization).

sembles the ground truth distribution of trip distance, meaning that the sample is *not* skewed towards larger values. This configuration is useful when analysts want to prioritize other parts of the dataset while ensuring that the distribution for the trip distance attribute is representative.

6.4.3 *Subdivision: Splitting the data into bins*

To construct useful chunks from this now sorted, but otherwise unstructured and therefore difficult-to-query data, we next define a scenario-specific structure on top of this list by splitting it up into bins. We express these bins as Set[OrderedList[Item]], where each list OrderedList[Item] is a section of the linearized data, keeping the order defined in the linearization step. Rather than already producing the chunks at this stage, this intermediate step instead constructs a "search structure" of the data that we in the next step then query for the most relevant data at a certain point in time. Essentially, we take a "divide-and-conquer" approach to the search problem for useful items, shifting from a global scope to the scope of smaller bins.

There are many ways in which one may subdivide the linearized data. The two extreme approaches are to put all data into one bin or every item into its own bin. The former can be used to model a sequential read, taking the first n items from the bucket at every chunk, while the latter allows to directly query the data, for instance to find the top-k largest values. For all cases in between, we need to define a metric for comparing consecutive items that subdivides the data in a desired manner. In Figure 19, we outline three strategies. One simple metric is to divide the data by *cardinality*, subdividing it in regular intervals so that every bin contains the same number of items. This strategy makes little assumptions about the underlying data, making it a good fit for cases where analysts are unfamiliar

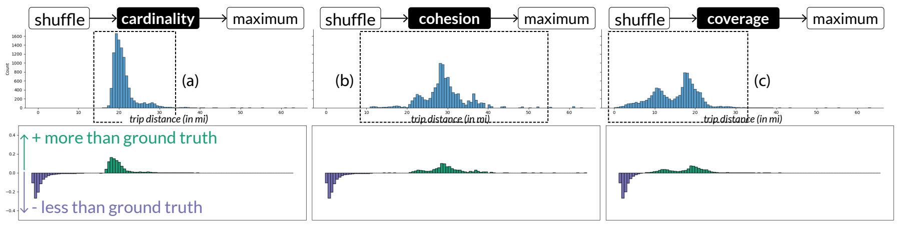


Figure 20: Impact of the subdivision strategy on the sampling ($||\mathtt{sample}|| = 10\mathtt{k}$). Depicted in the first row is the distribution along the trip distance attribute for subdivision by *cardinality* ($||\mathtt{bins}|| = 1,000$), by *cohesion* (splitting at the top $1,000$ biggest differences), and by *coverage*, with the latter two considering the trip distance attribute. The second row shows the relative difference in % compared to the distribution in the full dataset. We can clearly see all three samples being skewed towards larger values, with interesting differences in which values become most frequent: In the cardinality case, most values are found around $20\mathtt{mi}$ (a), cohesion around $30\mathtt{mi}$ (b), and coverage produces peaks at 10 and $20\mathtt{mi}$ (c).

with a dataset or where they want to explore it. For cases where the user interest is more clearly defined, we can, for example, increase the *cohesion* within bins by subdividing the linearized data whenever we measure a large difference between successive items. This provides us with a subdivision where each bin contains a set of similar items, which in turn allows us to tailor the sampling based on that similarity metric.

Yet, measuring the similarity may not always be possible, desirable, or useful and so another strategy is to increase the *coverage* over an attribute of interest. This means that, rather than making the items similar to each other per bin, we create bins that have similar statistic properties. This allows controlling for the probability of selecting certain values from a bin. In the example in Figure 19, we create bins for every successive pair of min and max values along an attribute in the data.

EXAMPLE: Exchanging the subdivision strategy has a noticeable effect on the distribution in the samples we draw, as depicted in Figure 20. Dividing the data by *cardinality* (here $||\mathtt{bins}|| = 1\mathtt{k}$), subdivides the data into bins that evenly subdivides the linearized (i.e., randomly shuffled) data. Thus, a "dense" region in the input where the linearization metric finds many similar values are spread out over many bins, while "sparse" regions are spread over few. The linearization used in the base pipeline, however, randomly distributes the data, so each bin potentially contains the entire value range for all attributes. Since we are selecting the maximum value from each bin, the distribution depicted in the figure is skewed towards larger values for this case.

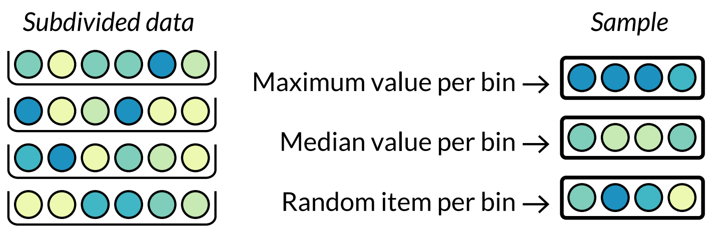


Figure 21: Exemplar selection strategies used on on the same subdivided data: Maximum value, median value, and selecting a random element.

Looking at the *cohesion* strategy in the second plot, we can see how the distribution differs from the first. This strategy further skews the sample towards larger values, because we can utilize two characteristics of our scenario: the trip distance attribute contains relatively few large values, and these values are randomly distributed because of the shuffling linearization. As a result, large values are likely to lie close to smaller values in the linearized data, and therefore, this subdivision creates new bins for the largest values in the data. As a result, this configuration allows analyst to prioritize extreme values.

Likewise, the distribution yielded by the *coverage* strategy in the third plot is also skewed towards larger values. Bins here are created by matching pairs of successive upper and lower values (0.05 and 0.95 quantile), which means that the rare, large values are also likely to appear in many bins, and therefore are likely to appear in the sample. As a result, the two samples yielded by the cohesion and coverage strategies in this configuration are rather similar.

6.4.4 *Selection: Placing items into chunks*

In the third and last step, we then construct chunks by selecting the most relevant items from the subdivided data using a scenario-specific prioritization strategy. Following the high-level input and output formats of progressive sampling, this step outputs the chunk format `OrderedList[Set[Item]]`. In the selection step, the goal is to construct a useful sample of the entire dataset by selecting the most appropriate items per bin from the subdivided data.

Given what data analysts are interested in, they can choose an appropriate selection strategy. Three examples are depicted in Figure 21. Analysts interested in extreme values, for example, may choose the *maximum* strategy, which selects the largest value along an attribute of interest. Analysts who want to get an overview over the dataset may select the *median* element for each bin to get a useful element at each chunk, or select items randomly to increase the spread of the data. For cases where the user interest is not clear, yet, analysts can

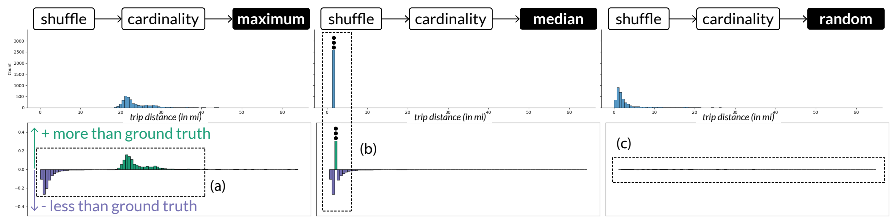


Figure 22: Impact of the selection strategy on the sampling ($||\mathtt{sample}|| = 10k$). Depicted in the first row is the distribution along the trip distance attribute for selecting the *maximum value*, the *median value*, and selecting *randomly*. The second row shows the relative difference in % compared to the distribution in the full dataset. We can clearly see how the selection step affects the sample: The maximum selection skews the sample towards larger values (a), while the median strategy selects mostly small values (b), since most taxi rides in the data are short distance. The sample produced with random selection mostly stays true to original distribution (c).

use the *random* strategy, which does not consider the values of the data but picks elements randomly from each bin.

How many items to select per chunk also depends on the analysis scenario. One heuristic is to select as many items as possible, while still ensuring that the progressive computation produces results within interactive response times [41], which can even be dynamically adjust to account for fluctuations in recent computation steps [110]. Another option is to always select a fixed number of items per bin, thus guaranteeing a fixed chunk size in the computation.

EXAMPLE: When comparing the effects of changing the selection strategy on the base pipeline, we notice clear differences in the distribution of the trip distance attribute in the plots in Figure 22. Overall, the selection strategy arguably provides the most "direct" way of controlling the output distribution in our example: Selecting the *maximum* value skews the distribution of the trip distance attribute in the sample towards larger values, selecting the *median* prioritizes average items (since a vast majority of taxi rides in New York City are relatively short distance, this strategy yields mostly short trips), and picking *randomly* means that we approximate the global distribution of the attribute. Thus, depending on the task, we can find clear use for all three: When looking for extreme values, analysts should choose a min/max strategy, while analysts interested in getting a representative sample may either select representative points from each bin using the median strategy, or get a representative distribution instead by selecting randomly.

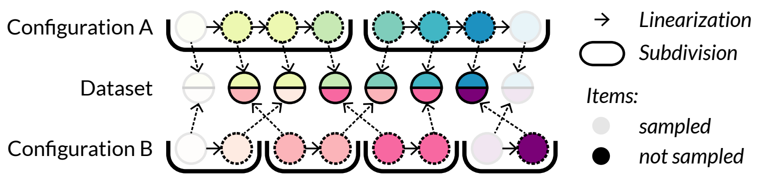


Figure 23: By storing pointers to the original input data, the pipeline configuration can be tailored at runtime using precomputed linearization and subdivision structures.

6.5 TAILORING PVA-SAMPLING ON-THE-FLY

The second challenge for PVA-sampling we outlined in the Introduction is that PVA is an inherently dynamic process, in that the analysis task may change while the computation is ongoing, the input dataset may grow over time, or analysts may want to prioritize interesting subspaces of the data. Rather than having to restart the analysis to adjust the sampling (and thereby breaking the flow of the analysis [33]), the sampling should be tailorable dynamically. In this section, we demonstrate how the modular architecture of the pipeline allows us to just that, showing how it accounts for changes in task, dynamic input data, and changes in scope.

6.5.1 *Dynamically tailoring to changing tasks*

In contrast to non-progressive VA, the task a user performs on the data in PVA may change mid-analysis, as new insights arise from the partial results. Analysts in the *progressive explorer* role in particular, who use PVA to "gain a comprehensive understanding of the data and process" [78] may repeatedly switch their task.

The impact that changing the task has on the sampling is evident, as the task influences the pipeline's configuration as different parts of the data become most useful. Dynamically changing the task effectively means that we need to be able to exchange any parts of the pipeline at any point in time. The challenge here is that – as with the long-running computation on the data that is made interactive by the sampling in the first place – the complexity of the data increases the complexity of processing the pipeline steps.

Both the linearization and subdivision step access the entire dataset, and depending on the complexity of the chosen strategy, exchanging (recomputing) them can cause noticeable delays in the analysis. One way to nevertheless enable dynamic exchange is by precomputing combinations of linearization and subdivision operations, thus paying the computation cost ahead of the analysis. To account for changes in the analysis scenario at runtime, we can then simply use either of the precomputed data structures as input to the selection step (see Figure 23). For example, by storing pointers to the input dataset,

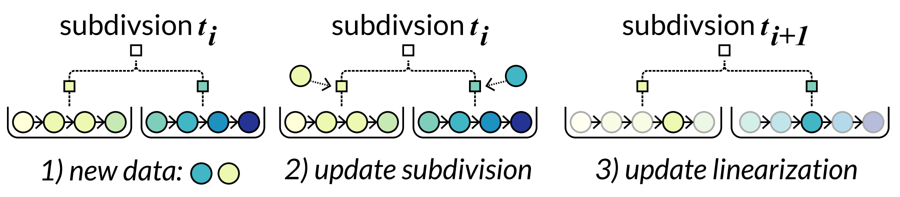


Figure 24: An incremental variant of the pipeline concept, showing how using dynamic data structures for representing linearized and subdivided data allows including new data to the pipeline, thus allowing to incrementally update the sampling on-the-fly without accessing the entire dataset.

we can keep track of which data has been sampled so far, even when exchanging the subdivision. To be economically viable, however, analysts need to consider how much precomputation is "worth it". When gaining a first overview for a new, unfamiliar dataset, this preprocessing may in fact *not* be worth the wait, yet for these cases, relying on one pipeline may suffice. For analysts working in a professional context, on the other hand, who often analyze the same dataset multiple times, having a catalog of linearizations and subdivisions to tailor their sampling can be particularly useful.

The selection step on the other hand can be evaluated on a per-chunk basis. Rather than naïvely running it exhaustively over the entire dataset to preemptively set a chunk for each item, we can instead construct the next chunk on-demand, significantly reducing its complexity. That way, we essentially use the selection step as a query over the subdivided data, allowing for efficient on-demand retrieval. It also means that we can dynamically exchange the selection operation at runtime by simply changing what query we use here.

6.5.2 *Dynamically tailoring to incremental input data*

Next, we look at how the pipeline can accommodate changes in the input dataset. Up until now, we took an "upstream", global perspective on PVA-sampling, where sampling is positioned at the beginning of the PVA process with access to the full dataset, and where all other operators in the PVA process wait for the chunks produced by it. This perspective is also rather common in the literature [68, 107]. However, as noted by Schulz et al. [102], it falls short in capturing dynamics in PVA. This is because different operators along the PVA process have different requirements to the input data, with some operators like clustering requiring broad data than a progressive scatter plot. Thus, using only one sampling ahead of the process is insufficient. The solution proposed by Schulz et al. is a buffer/sequencer model for incremental visualization, where each operator gets to manage their own priority queue. To integrate dynamic input data with the pipeline, we essentially need to make the sampling itself incremental, such that whenever the input of dataset of the pipeline changes, the chunking can reflect that as well. In other words, we need to incrementally compute the linearization, subdivision, and selection steps.

To make the first two steps of the pipeline incremental, we can simply maintain incremental data structures for linearized and subdivided data, into which we can insert new and from which we can remove processed data. For example, for an incremental linearization of tabular data, we can use a sorted index structure like a binary search tree over List[Item]. Whenever the input data changes, we can efficiently remove or insert those items using that index, and in effect make the linearization incremental. Similarly, we can adopt incremental subdivisions by using data structures like incremental segment trees [123]. A tree structure is compatible with Set[OrderedList[Item]], in that leaf Items are grouped by nodes of the tree, and this structure can be also efficiently updated incrementally. For complex subdivision operations like 1-dimensional clustering, dedicated incremental algorithms can be utilized, which are discussed in the database community [103]. Lastly, as discussed in the previous section, the selection step already can be run on-demand per chunk, rather than exhaustively over the entire dataset, thus it is inherently fit for incremental updates.

6.5.3 *Dynamically tailoring to changing scope*

A big advantage when progressively analyzing data compared to analyzing it in one step is that analysts can steer the computation towards data subspaces that currently interest them, while that computation is still ongoing. In other words, the scope of the analysis scenario can change dynamically (from the entire dataset to a subspace of interest). Steering generally means to prioritize data inside a user-selected region of interest in the sampling while the analysis is running, retrieving other data later [53]. As a result, the visualization of that subspace is “completed” earlier, allowing for more certain decisions on the data, making steering a powerful mechanism for rapidly exploring emerging patterns in the visualization. An example of this is a *progressive searcher* [78] zooming into a visualization to see details about the region that interests them: The computation can then focus on data that lies inside the zoomed-in region, rather than spending resources on data that lies outside of it. Below, we show how steering can be integrated into the pipeline (see also Figure 25).

Chunking during steering differs from regular sampling in multiple ways. One difference is that chunks during steering are no longer samples of the entire dataset, but they are instead skewed by the subspace of interest. When steering the progression, the chunks analysts see may exclusively contain data from that subspace, while after the steering, i.e., once that subspace is exhausted, items from

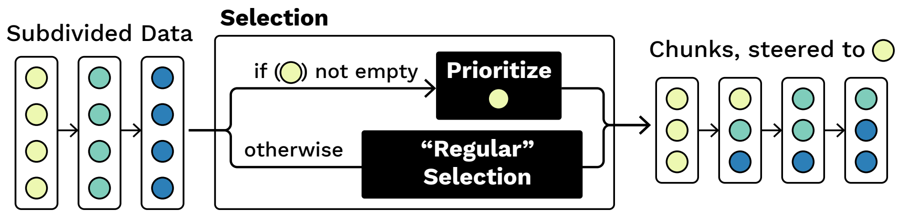


Figure 25: Integrating computational steering with the sampling pipeline: Computational steering can be expressed as an extension of the selection step. This selection prioritizes a subspace of interest by moving those data to the front of the chunking and then applies the "regular" selection strategy on the remaining data.

that subspace will not appear in chunks at all. Another difference is that steering is situation-dependent, in that interesting subspaces arise *while* the computation is running, and analysts often change what subspace they steer towards multiple times in the same progression (see the progressive observer role [78]). As a result, steering can rarely be configured before the sampling starts, unlike the conditions for representative sampling.

In the context of PVA, it is therefore necessary to be able to model steering as part of the pipeline, to leverage the full potential of PVA. We can do so by considering the data format steering operates on. Effectively, steering reorders the items in the chunking, such that items from the subspace of interest appear in early chunks, while the remainder of the data is sampled afterwards. In terms of the data format in the pipeline, steering is, thus, a transformation with the format $\mathrm{OrderedList[Set[Item]]} \to \mathrm{OrderedList[Set[Item]]}$. This means that we can model steering as a substep of the selection, such that we can integrate it with any existing selection operation.

One way to achieve this integration is to focus the selection step on a single or at least a subset of bins that contain interesting data. A requirement here is, however, that the data is linearized and subdivided by a suitable similarity metric, such that bins group data that are similar, such that interesting items appear in similar bins. In our running example from Section 6.4, for instance, an analyst may be interested in taxi rides that take place around midnight. If the data is subdivided into hourly intervals, we can steer the progression by selecting items only from the respective bins from that interval.

In cases where the similarity metric does not match the user interest, we can integrate steering by adjusting the selection strategy used for each bin. For example, to prioritize taxi rides around a certain time of day over all bins, we can select items from that interval for bins that contain such items, and otherwise select items that happened as close as possible to that interval. Thus, we need to essentially adjust the selection strategy *per bin*, based on what data it contains. To achieve this steering, we can maintain simple descriptive metrics for each bin

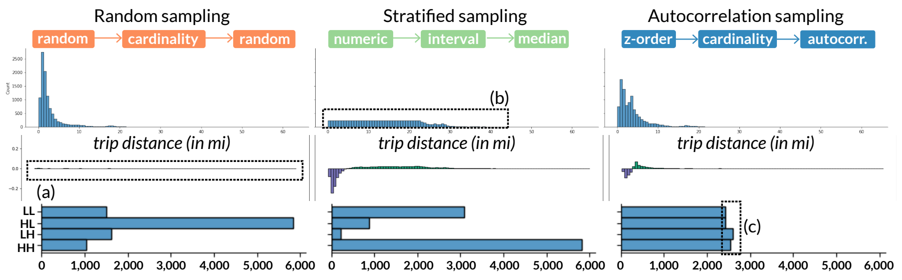


Figure 26: Examples of existing sampling methods recreated using the pipeline: Random sampling, stratified sampling, and spatial autocorrelation sampling. (a) Random sampling maintains the distribution along the trip distance attribute, while stratified sampling purposefully samples the value range evenly (b). Autocorrelation sampling on the other hand ensures that the autocorrelation categories are equally distributed in the sample (c).

(such as min/max/mean per attribute) and then switch the selection strategy for each.

An even more dynamic option to integrate steering is to combine the previous two approaches, adjusting the number of items selected per bin, based on whether it contains items or not (as outlined in Figure 25): Then, the selection greedily selects all items from the subspace of interest from all bins, until the chunk is "full" or the subspace is exhausted. Again, this can be achieved rather straightforward manner by maintaining descriptive metrics for all bins, this time basing the number of selected items on them.

Thus, depending on how accurate the steering should be and how much effort is viable, the sampling pipeline can be fit to these needs. As a result, any tailored sampling pipeline following the input and output formats outlined in Sec. 6.4 can benefit from steering. Second, we can vice-versa integrate any steering mechanism with the pipeline, as long as it supports the format of the selection step. This means that, regardless of how exactly a subspace is prioritized (e.g., based on a one-to-one mapping [27], iterative rebinning [118], derived from decision trees [53]), it can be combined with the sampling pipeline through the query filter it defines over the remaining data.

6.6 Utilizing the Pipeline's Modularity for Tailored Sampling

Having formalized task-tailorable PVA-sampling into a modular pipeline, we next provide examples of how this pipeline can be used to benefit analysts, again considering the taxi dataset for reference.

6.6.1 *Recreating existing samplings*

By formalizing PVA-sampling along a pipeline our goal is not to replace nor outperform scenario-specific samplings, but we want to supplement them. In the previous section, we showed how analysts can configure custom samplings with the pipeline. Here, we want to exemplify another advantage of the pipeline, which is that we can also use it to recreate existing sampling approaches in terms of linearization, subdivision, and selection strategies. The idea is that, whenever the qualities of a particular sampling algorithm are required, they cannot only be expressed in the pipeline format, but then also further tailored and adjusted, *because* they are in the pipeline format. To demonstrate this, we model well-known sampling algorithms using the pipeline, with their output depicted in Figure 26.

- **random sampling**: random shuffling linearization $\rightarrow$ cardinality-based subdivision $\rightarrow$ random selection.
- **stratified sampling**: numeric sort-by-attribute linearization $\rightarrow$ interval-based subdivision $\rightarrow$ median selection.
- **sampling for balanced spatial autocorrelation**: z-order linearization $\rightarrow$ cardinality-based subdivision $\rightarrow$ balancing autocorrelation selection.

This highlights expressiveness of the sampling pipeline, in that we can create both simple approaches like random sampling, but also rather specialized approaches as in the spatial autocorrelation example. This sampling controls the distribution of four categories (called LL, HL, LH, and HH), which express whether a local value is greater than neighboring values (yielding H* or L* categories) and the global mean value (yielding *H or *L categories) of a spatial variable. This is inspired by the approach by Zhou et al [132], who demonstrate that sampling using spatial autocorrelation allows for effective exploration of large (and therefore often cluttered) geospatial datasets.

6.6.2 *Recomposing sampling pipelines*

Modular design reduces implementation efforts by increasing reusability of partial solutions, in that we can compose the operators from existing sampling methods to create a new approach. For example, we can recompose operators from the pipelines in the previous section into a new sampling approach. In the example depicted in Figure 27, which is inspired by z-order sampling proposed by Zheng et al. [130], we use the z-order linearization from the autocorrelation sampling, a cohesion-based subdivision, and the median selection from stratified sampling. Linearization and subdivision both tailor the sampling towards the spatial location of the data points, in that the linearization places points in successive order if they are close to

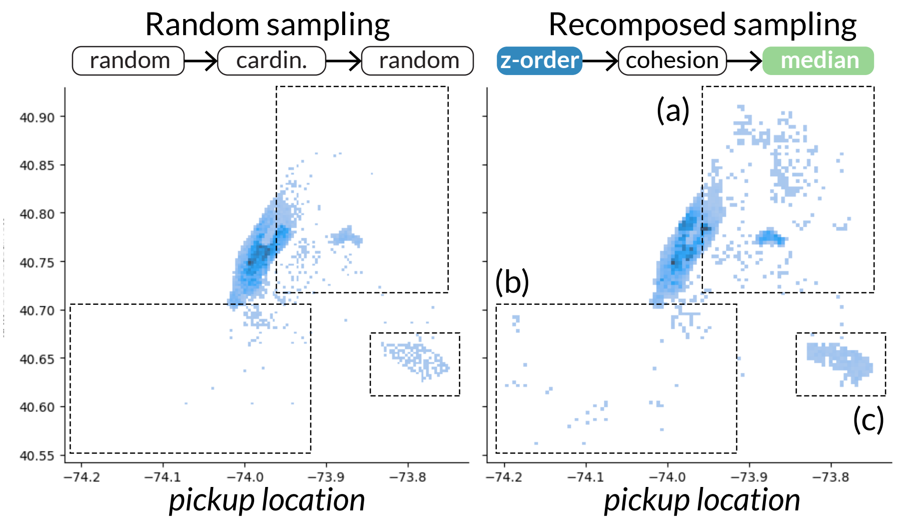


Figure 27: Example of tailored sampling created by recomposing the operators used in Figure 26 for a distinct sampling on the taxi dataset ($\|\text{sample}\| = 10\text{k}$). The tailored sampling depicted on the right noticeably preserves sparse regions (a), outliers (b), as well as local structures (c) compared to the random sample on the left.

each other in the pickup location attribute, and the subdivision splits up this data whenever there is a large distance between values. This results in bins that contain items from distinct regions in view space, and their cardinality depends on the spatial density of that region: densely populated areas are contained in large bins, while sparse regions are contained in small bins. In turn, when selecting elements from all bins, this increases the visibility of outlier points in the sample, while maintaining sparse regions as well as dense structures in the sample, which is clearly visible in Figure 27.

This demonstrates the modularity benefits of the pipeline, which allows reusing existing modules for tailored sampling rather than requiring a completely new implementation.

6.6.3 *Tailoring the sampling towards multiple attributes*

The modularity also allows to independently tailor each step of the pipeline to account for a different aspect of the analysis scenario, covering complex analysis scenarios. As an example, we here configure a sampling tailored for analysts exploring the relationship between the spatial distribution of long taxi rides in December of 2018. Accordingly, we configure a pipeline tailored towards three attributes at the same time: The linearization sorts the data along a z-order curve over the pickup location, the subdivision increases coverage over the trip distance attribute, and the selection picks the maximum value along the pickup time attribute. The linearization ensures that items within the same bin are also located close to each other in geospace, which

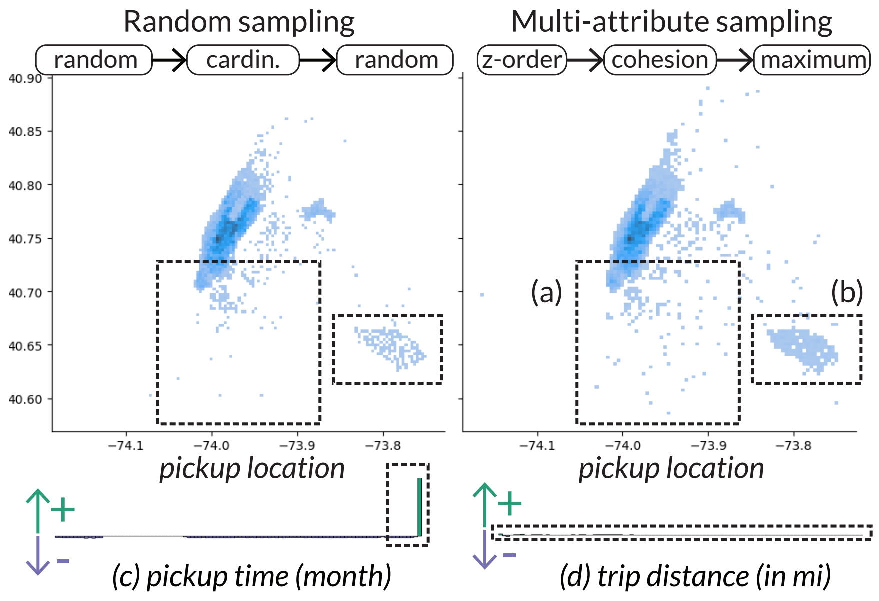


Figure 28: Example of sampling tailored to multiple attributes on the taxi dataset ($\|\mathtt{sample}\| = 10\text{k}$). The pickup location in the tailored sample better maintains both outliers (a) and dense regions (b) than a random sample, albeit not as clearly as in the sampling tailored *only* to the spatial distribution in Figure 27. The histograms show that the sampling is also clearly skewed to pickup times late in the year (c), while the distribution of the trip distance matches the overall distribution in the dataset (d).

helps to preserve outliers and dense regions in the sample when selecting items from all bins. The subdivision then ensures that all bins contain both long and short trips, which ensures that this distribution is maintained in the sample. The selection then skews the sample towards trips that are latest in the year. The effect is visible in Figure 28.

This example demonstrates the flexibility of the pipeline in two ways: First, we can consider multiple data *attributes* in the sampling. This means that the sampling can be tailored to address some of the more complex analysis scenarios, controlling the sample distributions for more than one attribute. The second benefit is that we can consider multiple data *types* in the sampling. This is noteworthy, as many existing sampling algorithms are geared towards a particular data type (sampling for spatial data, temporal sampling, sampling multivariate data, ...).

6.7 LIMITATIONS

Since the pipeline works on the general PVA-sampling input and output format that inputs an arbitrary structure and transforms it to chunks of items from that structure, we can (at least in theory) support a wide range of analysis scenarios: As long as we can find a linearization algorithm for the data structure, the data can be transformed into chunks along the pipeline steps. Nevertheless, this flexibility brings some caveats, which we discuss below.

One limitation is that, because of its generality, the pipeline cannot formally guarantee optimal performance across scenarios, meaning that in cases where the sampling cannot be expressed along the three steps of the pipeline, a dedicated implementation may be required. For example, when the qualities of an existing PVA-sampling (see Section 6.3.3) are needed, it can make sense to rely on these "off-the-shelf", optimized solutions, rather than recreating them with the pipeline steps. This poses a trade-off, though, as using tailored, task-specific solutions may come at the cost of losing future tailorability, once the task changes. Its generality also means that the pipeline cannot guarantee usefulness: Analysts need to carefully consider the strategies and parameters used at the three steps in context of the task they want to support, as the sampling may otherwise be inefficient. In our running example, for instance, we configured the pipeline to skew the sampling towards maximum values along the trip distance attribute. This tailored sampling is only *useful*, if these long-distance trips are actually relevant to the task. As with any task-tailored PVA-sampling approaches, the pipeline requires analysts to actively choose a sampling that is useful to their analysis, but in contrast to those existing approaches, analysts need to do the tailoring themselves.

In the same spirit, another consideration before using the pipeline is whether it makes sense to invest time and effort into tailoring the sampling, rather than using a sampling that is slightly subpar. One aspect to this economical question is the task that analysts perform on the data, and in particular how critical it is that chunks are tailored to the task. After all, for analysts casually visualizing a dataset to develop a first impression, random sampling might suffice, even if that is not the best fit for their data. Yet, cognitive biases like the uncertainty bias ("misjudging the uncertainty of intermediate results" [94]) or illusion bias ("reading something into incomplete results that is not there" [94]) as studied by Procopio et al. [94] are an inherent challenge to PVA. Reducing their impact is "part of the deal" when using PVA over non-progressive analytics in any analysis task beyond casual observation. The sampling pipeline, then, provides a flexible tool to that end, helping to reduce the impact of biasing analysts at the root of the PVA process.

6.8 conclusion and future work

In this paper, we introduced the notion of tailorable PVA-sampling, which differs from "regular" sampling in VA, in that it is a continuous process rather than a computation step, leading to a unique set of requirements. Tailorable PVA-sampling allows to fit the sampling mechanism to the task, to make the progressive visualization as useful as possible as early as possible. We achieved this by providing a pipeline consisting of three consecutive modules (linearization, subdivision, and selection), which allow tailoring the data distributions in chunking to fit the needs of the analyst. We demonstrated the flexibility of this pipeline in a series of examples, both taking a step-by-step perspective, where we demonstrate how exchanging each module affects the output, but also by taking a holistic view, showing how the modularity allows recreating existing sampling methods, reusing operators, and tailoring to complex user interests at once. We then showed how the pipeline can be dynamically exchanged at runtime, to account for highly dynamic user interests common to PVA, changes in the input data, and the scope of the analysis. Our approach allows, for the first time, to tailor PVA-sampling to the needs of the analyst without requiring dedicated reimplementations, while also allowing to adjust the sampling on-the-fly without restarting the analysis.

A next step for tailorable PVA-sampling is to provide tool support to make our conceptual work widely usable. To produce the figures in this paper, we used our Python-based proof-of-concept implementation ProSample (see Figure 6.4.1 for more details). In the future, we want to make use of our experiences along this early implementation and introduce additional requirements like performance, scalability, and wider applicability into our design. This will require careful consideration for choice of implementation language and API design to future-proof and integrate our pipeline with other frameworks. While tool support in PVA remains sparse, general-purpose frameworks like ProgressiVis [41] are on the horizon. Therefore, integrating a future PVA-sampling library with these approaches will not only help make tailored sampling more widely available, but also to make PVA itself more usable in general.

ACKNOWLEDGEMENTS

The authors wish to thank Jakob Burkhardt for his help with the implementation in early stages of the project. We gratefully acknowledge funding of this research by the Innovation Fund Denmark (IFD) through the Grand Solution project *Hospital@Night*.

7 STRATEGIES FOR ENABLING DEGREE-OF-INTEREST FUNCTIONS FOR PROGRESSIVE VISUALIZATION

Marius Hogräfer, Aarhus University, Denmark
Dominik Moritz, Carnegie Mellon University, United States
Adam Perer, Carnegie Mellon University, United States
Hans-Jörg Schulz, Aarhus University, Denmark

In preparation for submission to EuroVis 2023.

ABSTRACT

Degree of interest functions are an important tool for interactive visual analysis, as they express the relevance of data to analysts' tasks. However, degree of interest functions currently are only being used on relatively small datasets that can be processed in one step. For larger data that require a progressive, chunk-by-chunk processing, current degree of interest computations are of limited use, as the interest values can only be computed within each chunk and are thus not valid for the scope of all data seen so far. As a result, common visualization mechanisms building on top of degree of interest computations – such as visual guidance towards data of interest, preferential labeling of high-interest visual objects, or information hiding of uninteresting data – can also not be employed to their fullest extent, posing a major limitation to progressive visual analysis. In this paper, we address this limitation by proposing strategies that increase the scope of interest values beyond the chunk they were computed in. These strategies reduce the error of progressively computing interest values over the data seen so far by providing context from the processed data to the next chunk, and by recomputing outdated interest values. In a series of benchmarks, we show these strategies outperforming regular, chunk-based computation in terms of the error they make. Our results suggest selecting scenario-specific strategies to reduce the error and thus increase the utility of degree of interest computations on progressive visualization.

7.1 INTRODUCTION

Degree of Interest (DOI) functions are an important tool in the visual analysis toolbox. They express the relevance of each data item to the current user task relative to the rest of the data, which allows differentiating highly interesting from less interesting parts of the data. Once computed, these interest values can then be used to drive visual analysis methods like guidance [45], focus+context [50], or labelling [1]. In turn, DOI functions allow analysts to focus on those data that are actually of relevance to them, which is particularly useful when the dataset is complex (e.g.,, in size or dimensionality).

While they thus make it easier to analyze large datasets, DOI functions are currently not used in the context of progressive visualization, a relatively new visualization paradigm for large datasets [107]. In progressive visualization, the dataset is processed and visualized in chunks rather than in one go, allowing analysts to view early, incomplete visualizations their data. Through progressive visualization, it becomes possible to achieve responsive interaction latencies on large datasets and long-running computations, for which traditional approaches would take minutes or hours. As a result, analysts of progressive visualization have been shown to outperform analysts of traditional "monolithic" systems in terms of task completion time, while being just as effective in producing insights [127]. Progressive visualization and DOI functions thus share the goal of facilitating the interactive visual analysis of large datasets.

One reason why the two are currently not used together is that when computing the interest value for individual items in the context of the rest of the data, DOI functions require the dataset to be available all at once, while in progressive visualization, the data is only available in chunks. Due to this limited context, DOI functions inherently produce misleading results. An item that is interesting in a small chunk may in the context of the entire dataset not be interesting for the task at all, and vice-versa, and thus, interest values computed over chunks generally differ from interest values computed over the entire dataset. Due to this disparity, any analytic method enabled by DOI functions will also produce a misleading outcome, inhibiting the analysis: guidance for example may lead the user to data that in the global scope of the dataset is not interesting, and focus+context may erroneously hide interesting data. Another reason why the two are not yet used in symbiosis is that DOI functions are highly scenario-dependent, and analysts may frequently adjust the parameters used by the DOI function based on new insights. In progressive visualization, this is a challenge, because this scenario-dependency means that precomputing DOI functions for all parameter combinations is generally unfeasible. At the same time, developing a dedicated, incremental algorithm that learns the structure of the dataset from chunks over time is also non-trivial and a time-consuming endeavor, and therefore also generally unfeasible. Thus, none of the analytic methods enabled by DOI functions are currently used in progressive visualization. This is a major limiting factor to the interactive visual analysis of complex data, since it is exactly this complexity that these analytic methods could help alleviate.

In this paper, we address this limitation by identifying the conceptual barriers that lead to errors in progressive DOI functions, and by proposing generally applicable strategies for overcoming them. In sum, we make three contributions in this paper:

1. We formulate the research challenge of progressively visualizing results of context-dependent computations like DOI functions, which relate individual data items to the rest of the dataset, thus inherently producing errors when only parts of the data are available.
2. We propose conceptual strategies for reducing the error of DOI functions on progressive visualization by extending the context of these functions beyond the scope of the chunk, first by enhancing the DOI function's context with already processed items, and second by recomputing those items whose DOI values have become outdated with the arrival of new data.
3. We report on the results of a series of benchmarks evaluating these strategies, which suggest that by selecting scenario-specific strategies, we can reduce the error of DOI functions on progressive visualization.

7.2 RUNNING EXAMPLE

Throughout the paper, we will refer to a motivating example around a particular dataset and DOI function, that demonstrates the challenges of using DOI functions with progressive visualization.

As a dataset, we use the New York City taxi data, which for the year of 2018 contains data of around 77 million taxi rides in the city[1]. This dataset has three characteristics that are important in the context of this paper: (1) its size makes it a good candidate for progressive visualization, (2) its complexity in terms of dimensionality and its use of multiple data types means that analyzing it is not trivial [42], suggesting the need for supportive tools enabled by DOI function to facilitate the analysis, and (3) its rows are ordered temporally, which can lead to sorting bias when processing it in order, which means that what data is most interesting can change over time.

On that dataset, we use a DOI function that measures the "outlierness" of data items, processing and visualizing the data in chunks. Outlierness describes how irregular a data item is in the context of other items in the dataset, i.e., the more different that item is in relation to the overall distributions in the dataset, the higher its interest value. Finding outliers is a common analysis task on high-dimensional data like the taxi dataset, allowing analysts to detect

1 https://data.cityofnewyork.us/Transportation/2018-Yellow-Taxi-Trip-Data/t29m-gskq

items that do not follow the general trend of the data. It is also computationally complex, meaning that precomputing it over the entire dataset is not feasible. Moreover, what items are considered outliers depends on all other items in the dataset, which in turn makes it challenging for progressive visualization, where that data context changes over time. There are many ways of computing the outlierness including dedicated implementations for streaming data [128], yet for simplicity, in our example we will compute outlierness based on algorithms from the widely used machine learning library scikit-learn[2], in particular `Elliptic Envelope` (EE), `One Class SVM` (OCS), `Isolation Forest` (IF), and `Local Outlier Factor` (LOF). For our running example, we consolidate the outputs of multiple algorithms into an average value for a more complex outlierness score, as each algorithm produces a slightly different interpretation of what data to consider an outlier.

Specifically, we use the following DOI function based on the modular DOI function operators proposed by Abello et al. [1]:

$$DOI(d) = \sum_i \frac{1}{|i|} \times inter(comp_i(d)) \tag{1}$$

In this function, $comp_i$ refers to the scikit-learn implementations of the algorithms of each DOI computation, which are averaged. The interest function $inter(d) = 1/(1 + e^{-2})$ maps their outputs to a $[0, 1]$ range.

Computing that DOI function over the entire taxi dataset takes multiple hours, and therefore, it makes sense to instead process the data progressively in chunks, with new chunks arriving on average every 1.2 seconds enabling an interactive visual analysis of the data. Nevertheless, as the individual computations do not consider the full dataset, the interest values noticeably differ between the full and the chunked computation.

In response to this challenge, we present strategies for adjusting the DOI function such that a progressive interest computation can benefit the user. The remainder of this paper is structured as follows: First, we present the literature on related approaches, then we detail the conceptual challenges that introduce the disparate results in progressive DOI functions, followed by the strategies to resolve the issue. Lastly, we present benchmarks for evaluating the applicability of these strategies.

7.3 RELATED WORK

In this section, we situate our work within three related domains, namely degree of interest functions on complex data, incremental learning, and progressive visual analytics.

2 https://scikit-learn.org/

7.3.1 *Degree-of-Interest Functions*

DOI functions were first introduced by Furnas as a way to optimize the limited display space available at that time, to show those parts of a document that are most relevant to that part a reader currently works with [45]. The concept was later adapted to the visual analysis, for example by Heer et al. in DOITrees [50], a focus+context technique for prioritizing those nodes for display in large tree datasets that are currently relevant to the user, based on what nodes the user selected. There are some related approaches in the literature, where DOI functions are used in a somewhat similar fashion to what we use them in this work, in that either the dataset is large or changing dynamically. Van Ham and Perer, for example, adapt DOI functions on large graphs, utilizing an interactive text search to influence the interest of nodes [134]. Moreover, Abello et al. [1] present a modular design space for defining DOI functions over dynamic graphs, where the values assigned to one node in a graph differ when computing it along a temporal attribute. In these and other examples, the DOI function terminates fast enough for interactive updates, which means that a progressive approach and thus the optimization techniques we discuss here are not necessary.

In contrast to this and more similar to the progressive use case is the work by Hu et al. [57], who use DOI functions on streaming data, which similar to the progressive scenarios we discuss here arrives and is processed in chunks over time. To keep computation times low, their approach limits the DOI function to the *newest* data that has arrived, discarding data that is considered too old to be of relevance to the user. However, unlike streaming, in progressive visualization it is assumed that the chunks of data are part of a whole, finite set of items that accumulate, and thus we cannot generally exclude items from the computation based on their "age".

7.3.2 *Incremental Learning*

Another related field of research is incremental (or "online") learning, which looks at the algorithmic challenges of analyzing large datasets that do not fit into memory or where the computation is otherwise limited by complexity. Algorithms that are incremental approximate the result of running the original "batch" algorithm over the entire dataset, building up a partial model as more data becomes available, thus becoming increasingly more accurate in the predicted results. The field has produced many incremental variants for many common analytic problems, such as clustering [122], dimensionality reduction [118], and even outlier detection [128]. The conceptual challenges of incremental learning are similar to the challenges of running a DOI function over chunks of data, as we also need to approximate the result of a monolithic computation on partial data. Yet, each implementation "making" an algorithm incremental requires special consideration of the data structures and procedures of that algorithm, so to the best of our knowledge, no generalizing solution exists. DOI functions, however, are scenario-specific to support a particular task on a particular dataset, which requires specialized solutions, so progressive variants are generally not available for DOI functions. Moreover, even if an incremental implementation exists that can compute the DOI function in chunks in a way that ensures that is consistent with a computation over all data seen so far, we still need to update outdated interest values computed previously. In our work, we present strategies for minimizing the error when computing DOI functions in chunks, and solutions to update outdated values.

7.3.3 *Progressive Visual Analytics*

There also exists prior work dealing with progressive visualization of computations that are context-dependent like DOI functions, wherein the value computed for one item is relative to other data. Namely, these can be found in the field of Progressive Visual Analytics (PVA) [41, 107], which aims to use interactions on incomplete visualizations to control the parameters of the underlying algorithm. Examples of context-dependent computations in PVA are relatively sparse, and each implementation uses a dedicated unique solution to visualize results progressively. For instance, Badam et al. use a progressive visualization for t-sne dimensionality reduction [10] by rerunning that computation on individual chunks. Pezzotti et al. instead use t-sne iteratively by visualizing intermediate steps of the computation on all data, rather than by chunking the data [93]. Turkay et al. use an incremental version of PCA to show high-dimensional data [110] in a progressive scatter plot, training its model incrementally but projecting all data based on that model each time. Another example are aggregation functions from SQL as seen in ProReveal [61], which algorithmically allow updating the "global" aggregate by incorporating the intermediate result with the next chunk. In our work, we present solutions to enhancing the context of context-dependent computations, which can be used whenever no such dedicated implementation is available.

7.4 DEGREE-OF-INTEREST FUNCTIONS ON CHUNKED DATA

7.4.1 *Problem Description*

Next, we characterize the challenge that prevent us from using DOI functions in progressive computations.

The strategies we present in this paper specifically address those DOI functions that are *context-dependent*, i.e., the interest values put parts of the data in context with the rest. For context-dependent DOI functions, the same data item can thus both be of great or low interest to the current user task, given two different data contexts, as described in our running outlierness example in Section 7.2. As a consequence, interest values computed on a subset of the entire dataset often differ from interest values computed on the entire dataset. While this on one hand means that a DOI function produces misleading results when run in chunks, it also means on the other hand that even if we run the DOI function over all data that has been processed so far (i.e., essentially a bigger subset of the data), the interest values are still likely to differ from a computation over *all* data. To overcome this, we would essentially need to predict, what other data there will appear in the progression, which is at least an error-prone, if not futile endeavor (in that if we could predict future data accurately, we wouldn't need the progression). For a more data-driven approach, we rely on the expectation that the more data we process, the more similar to the full computation the result would get if we ran the DOI function over all data processed so far.

Therefore, the challenge that our strategies address is to ensure that interest values computed for each data item at every point in the progression are as similar as possible to the hypothetical interest value computed over the data processed so far.

7.4.2 *Solution Outline*

Here, we outline our approach for addressing the challenge described above. To increase the similarity between the interest values computed progressively in chunks and the hypothetical full computation, we consider two perspectives: On one hand, the DOI function for the next chunk requires context of the data that was processed before, and on the other hand, the interest values computed before requiring the context of the data in the next chunk. Since including all data in the DOI function can quickly lead to computation times that are no longer fast enough for interactive analysis, we need strategies for reducing the size of these contexts while guaranteeing useful results. To provide context to the DOI function of the next chunk, we present strategies that generate an appropriate representation of the data processed so far. Then, to introduce the new data into the result set, we present strategies for selecting those elements in the data that are outdated and need to be recomputed. To account for the scenario-specific needs of a particular DOI functions, we provide a catalog of strategies that apply different heuristics to select data items, thus providing a selection of options to choose from, rather than a one-size-fits-all solution. This general procedure is depicted in Figure 29. We would

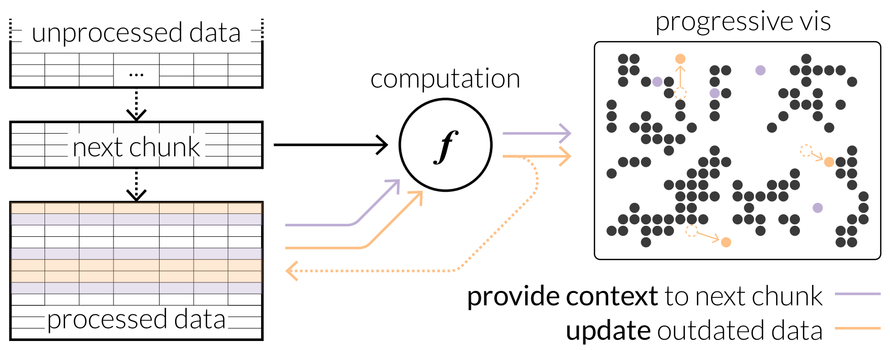


Figure 29: General approach for enabling DOI functions for progressive visualization: items from the processed data are selected as context for the computation, and "outdated" items are selected and their values recomputed in light of the latest data.

like to point out here that strategies addressing one part generally cannot be transferred to address the other, since they have inherently different goals: The context we provide to the next chunk should be representative of all processed items, while the outdated items do not inherently have to be. Nevertheless, there may exist some edge cases in which the two steps can be unified to increase the productivity of the DOI function, which we discuss separately.

7.5 ENABLING DOI FUNCTIONS FOR PROGRESSIVE VISUALIZATION

In this section, we present concrete strategies to implement the general strategies for enabling context-dependent computations for progressive visualization laid out in Section 7.4.2. We present the strategies in two parts: The first part is concerned with providing context to the computation of the next chunk, such that the new interest values approximate running the DOI function over all data seen so far. The second part is then concerned with ensuring that previously computed interest values remain consistent with the new data. We describe the general procedure behind each strategy and then discuss different approaches for implementing them.

7.5.1 *Strategies for Providing Context to DOI Functions in Progressive Visualization*

The first consideration is how to provide context to the DOI function for the next chunk to ensure consistency with previous data. As outlined in Section 7.4.2, our general approach here is to include a representation of the processed data as context.

A naïve approach is to include no processed data at all, which, while it maximizes the number of novel data processed at each step,

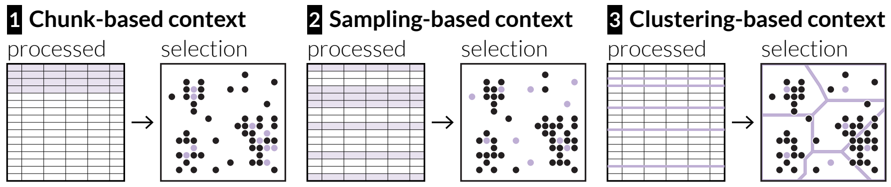


Figure 30: Strategies for selecting data as context for the next computation, at the example of a scatter plot. The colored dots in the scatter plots represent those parts of the processed data that are selected, while black dots represent those parts that are not. The first strategy is to select the oldest data at every step (1). To generate a representative set of items from *all* previous chunks, the processed data can be sampled (2). Lastly, representative points can be computed using clustering (3).

as outlined in Section 7.4.1 also leads to inconsistent interest values. The opposite naïve approach that in theory guarantees perfect consistency is to include all processed data in every DOI function. As this is generally not feasible, strategies are needed to determine what data best to include in the next computation to provide sufficient context to that computation. The strategies we present in this section are: *chunk-based* selection, in which one or multiple previous chunks are selected as context, *sampling*, in which a subset is drawn from the processed data, and *clustering*, in which representative candidates are computed using unsupervised learning (see Figure 30). The general trade off lies in the potential quality of the context a method yields and the required computational overhead to achieve it. We summarize considerations for choosing context strategies in Table 2.

chunk-based context The first strategy for selecting context data is to extend the DOI function with chunks retrieved previously. The idea is that instead of using a complex computation to find a representative subset, we can make use of the fact that the chunks themselves are (to a certain degree) representative subsets of the entire dataset. The chunk-based approaches below then differ in the way they select which chunks to include as context, namely the *most recent* or *random* chunks.

The first approach is to select the **most recent** chunks as context, under the assumption that those are the data most relevant to analysts. A major benefit of this approach lies in its conceptual simplicity. While all other context strategies require users to comprehend more complex selection algorithms, looking at the most recent data ensures that the progression simply considers data the user has most recently been introduced to. Moreover, this approach can be implemented rather efficiently into progressive systems, by using an appropriate variant of the `queue` data structure available in many programming languages. The drawback of this approach is that older data is not

Context Strategy	Considerations
Chunk-based	• Rather than revisiting already processed data, can the chunk size be increased instead? • Is the analysis task focused largely on the most recent data? Include the most recent data as context. • Is there a sorting bias in the data? Include older or random chunks to provide "historic context".
Sampling-based	• What sampling method is most beneficial to the computation? Can this method be achieved within the required response time? • Is there a data subspace, the analyst is most interested in? In that case, prioritize data from that subspace (i.e., steer the computation).
Clustering-based	• What groupings of data would supply the best context to the DOI function, and is there a clustering method that finds these patterns? • Does an incremental variant exist for this clustering method or do we need to recluster every time? • Can this clustering be provided within the required response time?

Table 2: Considerations for using context strategies.

part of the context, which can be problematic if those data are more relevant to the user task. The alternative is thus to select previous chunks **randomly** as context. That way, we can reduce ordering biases in the chunking at the cost of simplicity.

SAMPLING-BASED CONTEXT The next strategy for selecting context data is to draw a representative sample from the processed data. In contrast to chunk-based context, the sampling-based strategy generates context across chunks.

In this approach, we select a **representative** subset of the processed data that resembles its characteristics. The idea is that the DOI function then "sees" the same data characteristics in the next chunk as it would in the entire dataset. One way to draw a representative sample is to run random uniform sampling over the processed data at every computation step. This can, however, become computationally costly in later stages of the progression, as the processed dataset increases in size and may not fit into memory anymore. We can instead maintain a *progressive* random sample of the processed data, for instance using reservoir sampling [113]. Reservoir sampling guarantees that the probability of an item to appear in that progressive sample is at all times constant for all items seen so far, regardless of the chunk an item was processed in. This makes reservoir sampling a good fit to maintain a representative subset of the processed data in progressive visualization. However, for analysis tasks for which random uniform sampling would not be a good fit, neither is reservoir sampling, meaning that more specialized sampling methods are necessary. One example here is progressive pyramid-based sampling, which is specifically designed to preserve outliers particularly well [23].

CLUSTERING-BASED CONTEXT The third strategy for selecting context for the DOI function is to compute representative points using clustering. The main difference to the previous strategies is that the clustering-based strategy aims to find structures in the actual data values to get an optimal representation. However, the clustering-based strategy is more computationally demanding than the previous ones. We discuss two examples of the clustering-based context strategy, namely *representative-based clustering*, in which the context is extracted from points that best represent a part of the data, and *density-based clustering*, in which the clusters of similar density are sampled for context data.

The first approach is to compute the context using **representative-based clustering**, such that each context element represents the subset of items in the processed data with the closest Euclidean distance. While representative-based clustering algorithms like k-means are relatively simple, this approach, nevertheless, often produces "good-enough" interest values in many use cases. Losing et al. for example have shown how the k-means algorithm can be used to efficiently summarize the processed data, when computing with the k Nearest Neighbor classifier over streaming data [75]. Alternatively to k-means, other representative-based algorithms like k-medoids can be used, to overcome well-known shortcomings of k-means, for instance its sensibility towards outliers, which for some tasks can be detrimental.

An alternative approach is to use **density-based clustering**, where groups of similar data are found based on the data density. This better captures the structure of clusters with varying cardinality, since the representatives computed with the previous approach do not consider cluster sizes. This can be an issue when the data contains a highly dense region, since the clustering will produce representatives for other parts of the data as well that are essentially outliers, leading to a skewed context. To ensure that the most frequent trends in the data are also represented as such in the context, we can select the context as a sample of the clusters, with cluster size deciding on how many elements are sampled. Since this approach does not require the clustering to produce a set of representatives, other families of clustering algorithms are feasible alternatives. Many clustering algorithms exist, each with its own strengths in capturing different types of patterns in the data [122], covering which lies outside the scope of this work.

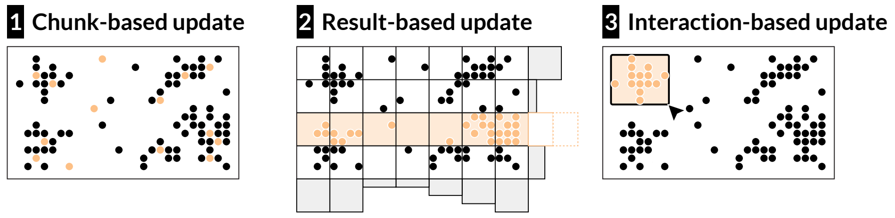


Figure 31: Strategies for selecting outdated data for update: One strategy is to select data based on a probabilistic metric, for instance with random uniform sampling (1). The second strategy is to base the selection on a metric in result space, for instance by selecting bins from view space with the most changes in the previous step (2). The third strategy is to use interactions made by analysts, for instance the last brush in view space, to select those data that are most relevant to the analysis task (3).

7.5.2 *Strategies for Updating Outdated Interest Values*

The second consideration is then to ensure that previously-computed interest values remain consistent with data that was retrieved later on. The strategies we present in this section, therefore, present different means of detecting outdated data (see Figure 31).

As with the context selection above, the naïve strategies to either update none or all interest values are not useful options here as well. Updating none of the previous data would lead to an inconsistent visualization that can mislead analysts, while updating all data quickly becomes unfeasible as too much data accumulates. Thus, dedicated strategies are necessary to select a suitable subset of the processed data to update it. Moreover, we cannot reuse the strategies we proposed in the first part above, since we want to specifically find those data for which the interest values are *outdated*, i.e., the subset we select is *not* representative. We present three general strategies: *chunk-driven* update, in which we recompute interest values based on their "age", *result-driven* update, in which we update those data for which the latest interest value lies in a certain value range, and *user-driven* update, in which we use knowledge about the user attention for selecting the most relevant data. We summarize considerations for choosing between the strategies in Table 3.

CHUNK-BASED UPDATE The first strategy for selecting outdated data for updating is to use its "age" as a heuristic, i.e., the last time they were updated by the computation. The general idea here is that interest values lose their validity over time as more and more new data accumulates, meaning that running the DOI function again on an "old" data item given the latest context will likely produce a different interest value. To achieve this strategy, the "age" of the interest values — meaning the last time they were computed — must be maintained, for example as a timestamp. This strategy prioritizes consistency, in

Update Strategy	Considerations
Chunk-based	• Do results of the DOI function become outdated per chunk or per item? If the entire chunk is likely to lose its validity, use this strategy. • How many times should the same chunk be updated, i.e., what is the maximum age per chunk?
Result-based	• Does the DOI function benefit from seeing data from all different value ranges? • Can results be binned progressively or should they be rebinned anew with every update? • How many items should be included per bin? Should it match the distribution observed in the results, or should the every bin be represented by the same number of items?
Interaction-based	• Are the interactions in view indicative of current analysis focus? • What items are affected by which interaction technique, and which of these items are currently the most relevant for the analysis? • What items outside that focused set should also be updated?

Table 3: Considerations for using update strategies.

that all data is kept equally "up to date", i.e., we can guarantee that all interest values are of a certain maximum age.

The first example for chunk-based update is to select data in a **first-in-first-out** manner, in that at every chunk, the data that has not been updated the longest is selected. The intuition here is to keep all data equally "up to date" by recomputing the interest values for those items for which the last computation lies the furthest in the path. By selecting "the oldest" chunk at every step i guarantees that interest values are at most $i/2$ steps old. While this approach is conceptually simple, it comes with the drawback that this maximum age increases linearly over time regarding the number of processed chunks. Therefore, an alternative is to update data in **regular intervals**. That way, we can set an upper limit to the interest value age based on that interval. The drawback is that the number of chunks that require an update identified by this approach equally grows linearly per chunk. Instead of updating all interest values in regular intervals, a pragmatic solution is to draw an as-big-as-possible random sample from all "update candidates".

An alternative to selecting data for update based on their chunk is to instead use a **probabilistic** approach. At every chunk, we select items randomly based on some probability distribution. If an item is not selected for update, its probability to be picked in the next chunk increases. While this ensures that all items are equally likely to be updated after a certain threshold, it requires additional data maintenance.

RESULT-BASED UPDATE The second strategy is to update those interest values, that lie within a certain value range that is deemed outdated. The motivation behind this approach is that interest values often shift locally, rather than uniformly over the entire dataset; for instance when a local region in the data crosses the density-threshold from being considered inliers to outliers, or a new greatest value replaces the old one. A benefit here is that we can specifically select those data whose interest values actually have become outdated. To achieve this, we divide the space of interest values into non-overlapping groups of items. Then, whenever a metric computed for a group exceeds a threshold, the group's items are selected for update. The challenge thus lies in finding appropriate metrics that trigger the update. There are different methods for computing these groups, which we distinguish by the type of data they best fit.

If the data is distributed uniformly and the value range is known upfront, we can divide the result space into **equal-ranged** bins and then assign each item to its bin. Under these assumptions, we can sample the data for including in the update by randomly selecting items from every bin at every computation step. If, however, the value ranges are not known upfront, the total range values change throughout the progression. Then, whenever new minimum or maximum values arrive, we can recompute the binning and include those items for update, for which the bin has changed.

If the data follows a non-uniform distribution, we can divide the data into **equal-sized** groups, such that every group contains the same amount of items. Then, when new items are assigned more frequently into one bin, we select the items in that bin.

Lastly, if the data does not apparently follow either distribution, we can run **clustering** over the ranges, in order to group the stored data based on similarity. A simple way to achieve this is k-means clustering [73], which divides the data such that every item in every group has the lowest Euclidean distance to one of k center points. Whenever an item changes the group it is assigned to, it becomes a candidate to be included in the next computation step.

INTERACTION-BASED UPDATE The third strategy is to use analysts' interactions as a heuristic for selecting data currently in the user's focus, to then prioritize those for regular updates. The motivation is to effectively *steer* the updates towards data the user pays attention to. As such, the interaction-based update strategy inherits benefits as well as drawbacks from computational steering, in that it potentially increases the efficiency of the analysis, since computational resources are focused on those data that are relevant to the analyst. However, this approach can also potentially cause analysts to overlook certain parts of the data they were not interacting with, as these are not updated. The approaches presented here differ in the way they determine, what data the user is currently focusing on.

One way is to use the **current selection** a user makes in view space as an indicator of their current focus. This is inspired by the Sherpa method presented by Cui et al. [27], where the data chunking is steered toward data that matches the filter defined by a brush in view space. Whenever no selection is active, the filter defined by any previously active selections slowly "decays" by increasing its bounds, until the remaining data is sampled randomly. As reported in the Sherpa paper, the benefit of this strategy is that analysts intuitively understand this principle, since their last selection is what the computation will focus on. A limitation is that only the latest selection is considered and previous interactions have no effect on the selection.

An alternative is thus to build a more comprehensive **model of the user attention** by considering also older interactions. The motivation here is that the analyst's interest evolves over time and cannot be captured by just considering the latest interaction. Additionally, modeling the user interest as interactions with data over time also allows us to detect data the analyst has *not* interacted with, yet, which can be similarly indicative of their interest. A pragmatic way to build a model of the user interest is to count the number of times, an interaction in view space has affected an item, and to rank the data by that count. Then, data with the highest (or lowest) scores are included in the next computation step. While using a broader picture of analysts' interest to inform the selection of context data, this strategy does come at the cost of introducing more complexity into the system, while also being less transparent to analysts.

7.6 BENCHMARKS

Following Munzner's nested model [83], we evaluated our algorithmic contribution with benchmarks. We evaluated the strategies across five datasets and four DOI functions, comparing them in terms of their accuracy and speed against two baselines. Given the number of test cases that this entails, we here report on the main takeaways from these experiments. All benchmarks were conducted on common laptop hardware (Intel Core i7 8550U with 1.8 to 4.0GHz, 16GB RAM, 512GB SSD) running Windows 10. The strategies were implemented in Python 3.9, using the pandas, numpy, and scikit-learn libraries for implementing the test cases. The code for the benchmarks is publicly available on GitHub[3].

3 https://github.com/vis-au/prointerest

7.6.1 *Setup*

TEST CASES We compare DOI functions using the strategies with two **baselines**: (1) only computing the DOI function on new items (referred to as *no-strategies*), and (2) computing the DOI function on new items, but increasing the chunk size to the number of items that the context strategy provides (referred to as *bigger-chunks*). The first baseline corresponds to "doing nothing", i.e., using the DOI function progressively in chunks without reducing the error. The second baseline corresponds to the state-of-the-art adjustment to the analysis pipeline for reducing the error of DOI functions whenever no dedicated algorithm exists, by simply increasing the number of items retrieved with each chunk.

We compared these baselines with our approach of using strategies (we refer to strategies by the labels in parentheses in later parts of this section). We evaluated the following strategies for **selecting context items**. To evaluate chunk-based context, we tested the *random chunk* (RC) strategy that randomly selects processed chunks as context for the next computation and *most recent chunk* (MRC) that selects chunks that were processed previously. To evaluate sampling-based context, we tested *random sampling* (RS), which uses random uniform sampling over the processed data, *DOI-binning* (DB), which divides the processed data into 10 bins along their interest values and then evenly selects items from these bins as context. We also tested the *clustering* (C) strategy, which computes a k-means clustering over the processed data and selects items from each cluster, with $k = 10$, to be comparable to the binning case.

For **selecting outdated items** from the processed data, we tested the chunk-based strategies *regular intervals* (RI) that updates items based on their chunk periodically, and *last chunks* (LC) in which we update the most recent chunks, *oldest chunks* (OC), in which we consequently update the chunks that have not been updated the longest. For result-based updates, we tested the *bin-based* strategy (BB), in which items are selected for update based on bins, using again 10 bins. Input-based update strategies were not evaluated, as they purposely limit the selection of items that are updated to regions analysts interact with, rather than covering the entire data space like the other strategies.

As parameters for computations using the strategies, we used a chunk size of $2,000$ items and retrieved $1,000$ context items per chunk. Computations using the *bigger-chunks* baseline thus used a chunk size of $3,000$ items, while *no-strategies* effectively used $2,000$ items, thus requiring different numbers of chunks to process the input data. We ran update strategies in an interval of 10 chunks, retrieving $3,000$ items each time.

TEST CONDITIONS Overall, we conducted our benchmarks on five datasets. As real-world dataset, we used the dataset of New York City cab rides (containing 7 numeric dimensions and about 77 million items). We also used three synthetic datasets of different Gaussian distributions, and one synthetic dataset containing sorted integer values ranging from 1 to 1 million. To reduce the runtime of our benchmarks due to the sheer number of test conditions, we reduced the per-benchmark complexity by only retrieving the first $12,000$ items per datasets.

The benchmarks used the following DOI functions: *Outlierness* (measuring how unique an item is compared to the rest of the data as per our running example in Section 7.2), *greatness* (the greater a value across n dimensions, the more interesting it is), and *denseness* (the more neighbors an item has in feature space, the more interesting it is). Each function computes interest values on a range between 0 and 1, where 0 is assigned to the least interesting item in a computation, and 1 to the most interesting item.

PERFORMANCE METRICS We assess the performance of each test case based on two metrics. The first metric expresses the error that the test case introduces into the analysis in terms of the computed value compared to running the same DOI function over the entire dataset. The closer that error to 0, the more accurate the prediction. The second metric is the computation time, which measures the time for processing the data. The lower this value, the better we consider the performance of a test case.

7.6.2 *Results*

Here, we discuss the main results of our benchmarks, depicted in Figure 32 and Figure 33. The code for reproducing our benchmarks can be found in our GitHub repository[3].

GENERAL OBSERVATIONS The first insight from our benchmarks is that no combination of strategies significantly outperforms either of the two baselines. Scenario-specific combinations of strategies can, however, help reduce the disparity between ground truth and chunk-based DOI values. For every test case, we can find at least one combination of context and update strategies, which reduces the DOI error in terms of median or standard deviation (see Figure 33). However, none of the strategy combinations was a “silver bullet” that always outperformed the rest, and instead the performance depended on the particular combination of DOI function and dataset. This makes sense, since the strategies serve as a heuristic for providing the best possible context to the DOI function, which depends on the particularities of that computation. We also observed how some combinations

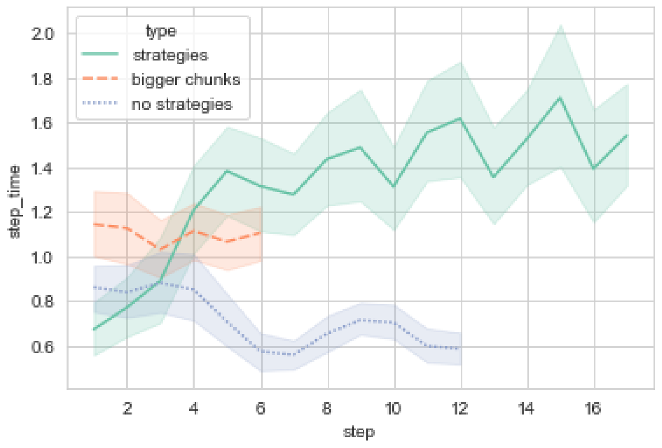


Figure 32: Summary of the runtime performance for the different strategies compared to the *bigger-chunks* and *no-strategies* baselines on the NYC taxi dataset. We can see that the test cases terminate at different points in time, since using bigger chunks means that the dataset is fully sampled earlier, while using the update strategies adds additional computation steps. We can also see the increase in runtime per step for the *bigger-chunks* and *strategies* test cases.

increased the error, for instance when using k-means-based clustering (C) as context strategy on a dataset that contained less than k clusters of data. Thus, picking the right strategies for enabling a DOI functions in progressive visualization requires careful consideration of the analysis task.

A second insight is that neither the strategies nor the two baselines can fully remove the difference to the ground truth DOI values to zero. As expected, we observed that when processing DOI functions in chunks, they always produce different results compared to computing that same function over the entire dataset, due to them being context-dependent (see Section 7.4.1). These differences are apparent both in the difference in the overall distribution of interest values after the computation completes, and the difference in the individual values produced per item in the data. Moreover, while in some cases the overall distribution computed in chunks exactly matches the distribution of the ground truth, the difference per item can be quite apparent. For example, when f=*greatness*, the per-chunk distribution exactly matches the ground truth, yet the values computed per item differ. This confirms that some DOI functions in progressive visualization produce an intrinsic error, which at the same time shows that evaluating chunk-based computations based on histograms of DOI distributions is an insufficient analysis tool for their quality.

Another insight from our benchmarks is that the strategies were able to alleviate sorting bias in the data (see results for the *sorted* dataset). Compared to *no-strategies*, most strategies produced better results in terms of median error and standard deviation in the f =

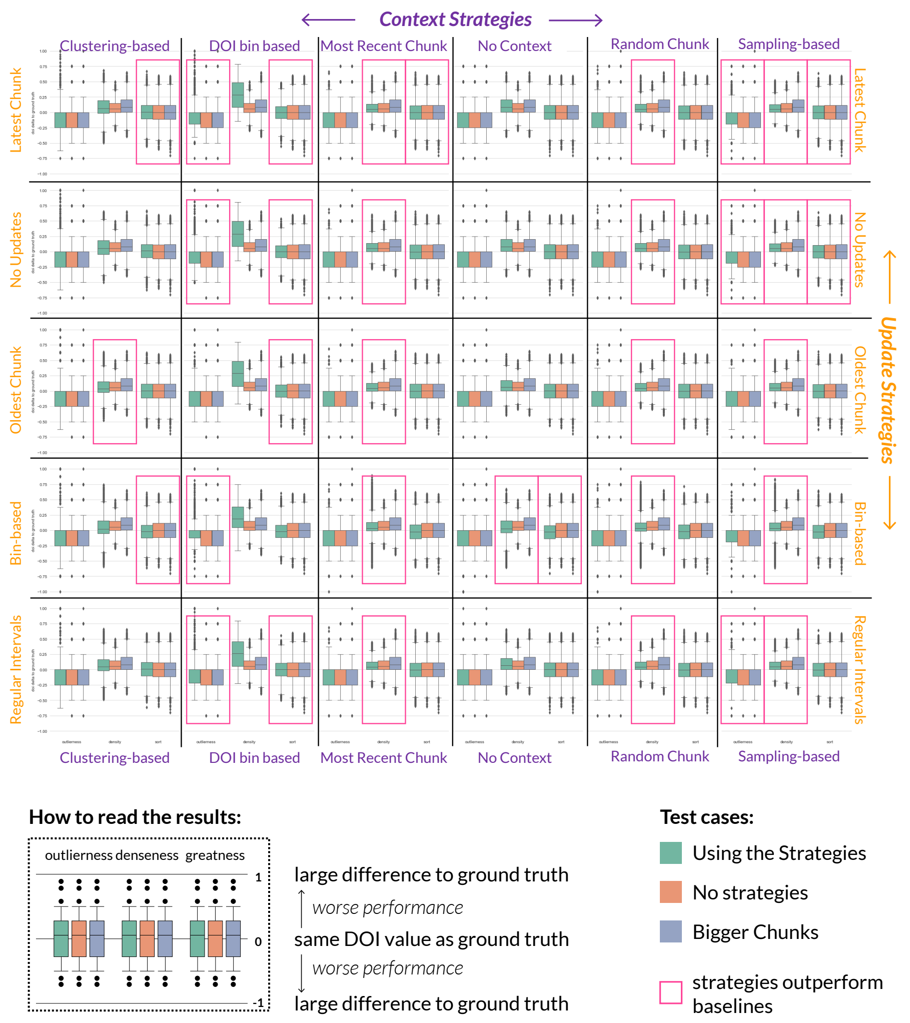


Figure 33: Benchmark results for the DOI error, showing the difference to the ground truth DOI values when running all three DOI functions on a sample of the NYC taxi dataset. Each cell in the matrix shows the error made by a particular combination of context (columns) and update (rows) strategies for all DOI functions, compared to the two baselines *no-strategies* and *bigger-chunks*. The closer a distribution around the 0 line in the center, the better. Highlighted in magenta are test cases, in which our strategies outperformed the baselines.

greatness, $D = \textit{sorted}$ case. This is due to the fact that providing context from past computations allow the computation to keep a "memory" of minimum values it has seen before, beyond the data in the chunk, and updating prior results allows incorporating knowledge about the new maximum. While the strategies thus helped reduce the error of this chunk-based DOI function, they did not fully remove it, for instance as not all values were updated after the final chunk.

A general assumption in progressive computations and an observation throughout our benchmarks is that increasing the number of items in a chunk-based DOI function increases the quality of the results, in that median error and standard deviation are reduced. This was evident in the improvements that using a context strategy case brings compared to *no-strategies*. Similarly, the results for *bigger-chunks* indicate that increasing the number of items that are processed per chunk in itself improves the computation, without the computational overhead of our context strategies. In fact, *bigger-chunks* outperforms some combinations of context and update strategies in our tests, yet for all combinations of datasets and DOI functions, there exist combinations of context and update strategies that further improve on *bigger-chunks*. The implication here is that when implementing a DOI function for progressive visualization, increasing the chunk size can already be a good solution to reduce the error, yet, by adding dedicated context and update strategies, the best results can be achieved.

DOI ERROR An overview of results regarding the DOI error is depicted in Figure 33. Some context strategies in our tests reduced the median error of chunk-based DOI functions and also reduced errors' standard deviation. However, the intensity of this effect for a strategy varied between datasets and DOI functions. The *MRC* context strategy in particular performed comparable to *bigger-chunks*, yet only processing $|item_{chunk}|$ new items per computation. Different context strategies did, however, show different variance in performance: Some consistently show small improvements over both baselines (*MRC, RS*), while other strategies in some cases show large improvements, but in other cases also diminishing results (*C*).

Update strategies also reduced the error of chunk-based DOI functions and consistently outperformed *no-strategies*. However, their effects in most cases mostly affected the standard deviation of errors, and where not as noticeable as context strategies in terms of the median error. Update strategies were most effective in cases where the dataset was sorted, in which case they often outperformed *bigger-chunks* in terms of error median and standard deviation. The *OC* strategy in particular performed well on sorted data. As with context strategies, update strategies' performance also depended on the dataset and DOI function and no strategy consistently outperformed others.

These observations again reflect that, just as the DOI functions themselves are designed for particular scenarios, so must be the choice of strategies used to reduce their disparity to the ground truth.

RUNTIME An overview of results regarding the DOI error is depicted in Figure 32.

The three test cases completed at different points in time. This is simply due to the different number of items processed at each step. Accordingly, the *bigger-chunks* case required the lowest number of steps (6), followed by *no-strategies* (12). Since update strategies were run as separate computation steps to the computation of new chunks every other step, test cases using the strategies needed the most steps (17).

Both context and update strategies also noticeably increased the per-chunk computation time over both baselines (see Figure 32). There are three main factors causing that increase: an increased computation time of the DOI function due to larger input compared to *no-strategies*, additional accesses to the database, and the additional computational overhead for computing each strategy.

7.6.3 *Threats to Validity*

In the benchmarks, we evaluated the strategies we proposed over a set of datasets, parameters, and degree of interest functions. Naturally, there are some limitations in terms of generality of the results. First, our tests were run exclusively on numeric, multivariate data, which means that we cannot draw any conclusions regarding categorical data, graphs, or temporal data. Moreover, in order to keep the total number of test cases manageable, our benchmarks are limited to a few representative implementations of the strategies we discussed in Section 7.5, which, nevertheless, resulted in many strategy combinations. This means that some of the strategies we proposed were not included in our benchmarks, and at the same time, not all potential variations of each strategy were implemented. For instance, for the clustering-based context strategy C we chose the k-means algorithm to generate groups of data for its conceptual simplicity, in lieu of methods like density-based or subspace clustering that may have performed differently in the tests. Lastly, our benchmarks were run with a fixed set of input parameters to the algorithms used in each strategy, for instance all bin-based methods (DB, BB) used a fixed number of 10 bins for simplicity. Thus, there exists some potential for tweaking the performance of the presented strategies, which we purposefully omitted to keep results comparable between benchmark configurations.

7.7 DISCUSSION AND FUTURE WORK

In this section, we critically reflect on the results of our evaluation and propose future research directions for DOI functions in progressive visualization.

7.7.1 *Increasing the Chunk Size vs. Using the Strategies*

A takeaway from the benchmarks is that increasing the chunk size *generally* improves the results of progressively computing the DOI function, while using context and update strategies improve *particular* use cases, but also add computational complexity. This seems like a clear limitation, and a natural question to ask is whether the strategies are at all practically useful, or whether one should just always increase the chunk size instead to get the best results. This notion places our strategies as direct competitors to sampling more data per chunk, yet we see them more as complementary enhancements. We discuss three ways in which our strategies can enhance the DOI computation below.

First, context strategies allow us to increase the scope of the DOI function, whenever we cannot increase the chunk size, as it is fixed by external factors. For example, when using DOI functions to inform the visualization of Progressive Visual Analytics (see Section 7.3), the chunk size is often dictated by the time it takes to retrieve and process the data with a complex analytic computation, as update rates should be kept within interactive intervals [41]. Thus, the scope of the progressive DOI function is limited by that chunk size, yet we can enhance that scope using context strategies. The runtime of the strategy then however also needs to match the interactive interval, for instance by dynamically adapting the context size using adaptive sampling [110].

Second, even when we can increase the chunk size, context strategies can nevertheless reduce the error the DOI function makes, assuming that the strategy fits the particular dataset and the DOI function. This is indicated in the benchmark results, where we found that no single "best" combination of strategies exists, yet there was always a combination of strategies that reduced the DOI error. In practice, this means that analysts need to consider, which items are best for improving the results of the DOI function, while also avoiding unwanted biasing. Analysts already consider, which data is of interest to them, when they define the DOI function for their progressive visualization, and considering how to provide appropriate context can be seen as an extension of that thought process. However, whether this process is intuitive and integrates well with the design process of progressive visualization requires further investigation.

Lastly, while increasing the chunk size and using an appropriate context strategy both aim at improving the DOI values of the next chunk, update strategies improve past errors. Therefore, they address a completely different challenge and can thus enhance the computation of DOI values independently of how new chunks are processed. Again, one needs to consider the characteristics of the DOI computation to select an appropriate update strategy, as well as when to update. In our benchmarks, we applied update strategies in regular intervals, yet that may not always be optimal. In fact, the more data we process, the less we can expect DOI values to change based on new data, so reducing the update interval may provide good results as well. More research is necessary to provide more detailed guidelines on how to choose and configure the strategies

7.7.2 *Applicability Limitations*

The strategies we propose in this work enable DOI functions for progressive visualization that relate single items to the rest of the data, which causes errors when computing these functions only over small chunks of the data. However, there are many ways to compute the interestingness of data items and not all computations are context-dependent, i.e., not all DOI functions relate individual items to the rest of the data. For instance, a simple DOI function is to compute the path length to the root node in a tree structure, or whether an item value inside a pre-defined range of interest. Such DOI functions that are not context-dependent do not benefit from the strategies we propose, as they already have perfect accuracy.

Another limitation of the strategies is that they introduce additional complexity into the visual analysis process, both in terms of computational overhead, but also conceptually. In some scenarios, it may simply not be "worth it" to wait longer and spend more time on selecting and implementing a particular strategy, as even incorrect interest values are helpful. When analysts simply want to get a rough, first picture of an unknown dataset for example, they may choose to accept the errors in their analysis, just to get to see the data as fast as possible. In other scenarios, even the erroneous interest values may be useful enough for their intended purposes. The strategies are the most beneficial, when the error in the chunk-based computation poses a risk to the success of the visual analysis. This again confirms that strategies must be selected purposefully for the particular analysis task, which means that using no strategies may be the best option.

7.7.3 *Using interest values in Progressive Visualization*

The focus of this work were the computational challenges in bringing DOI functions to progressive visualization, so a natural next step is to look at the other side of the coin, i.e., the challenges of actually using DOI function to support progressive visualization. A central difference to existing approaches is that interest values with the strategies change over time. DOI functions from their inception in visualization have been proposed as dynamic values, often consisting of a static *a priori* part that measures importance in the data and an *a posteriori* part that changes with user input [46]. However, using DOI functions progressively with the strategies means that the *a priori* part can change over time as well. This difference affects the way in which we can use the interest values in visualization, as we need to take into account this inherent uncertainty introduced by the progression. While recent work by Procopio et al. [94] suggests that non-expert users can effectively incorporate uncertainty into their analysis, it, nevertheless, harbors the potential for error. A potential solution could be to take into account the stability of the interest values, or as Angelini et al. put it, the "quality" of these values [5]. For example, when using interest values to enable guidance, in addition to evaluating the DOI function, only those values that have not changed in the last five iterations could be used. However, one observation we made in our benchmarks is that chunk-based DOI functions generally tend to overestimate the interestingness of items, meaning that it is exactly those interesting values that will change the most, as update strategies "fix" them over time. Further research is thus necessary on how to effectively make use of progressive DOI functions.

As a first step in this direction, we have implemented ProInterest, a progressive, interactive visualization tool that utilizes DOI functions based on our strategies. It is designed for progressively analyzing the NYC taxi dataset as discussed in our running example, based on web technologies. ProInterest will allow us to identify practical challenges for bringing degree of interest functions to progressive visualization in practice, and can serve as a platform to conduct user studies on. The code to ProInterest is available under open source licenses on our GitHub repository together with the code to our benchmarks.

7.8 CONCLUSION

In this paper, we discussed the challenges that arise when using DOI functions in progressive visualization. We formulated the research challenge of effectively updating context-dependent DOI functions for progressive visualization. We then discussed strategies for reducing the inherent error in chunk-based interest computations under considerations of the particular task and dataset of analysis scenarios. These strategies provide context to the next computation from previously processed data and find and update outdated interest values. In our benchmarks, we showed how scenario-appropriate strategies effectively reduce the error for different dataset and DOI functions. Our contributions enable further research in the use of degree of interest values in progressive scenarios, paving the way for analysis tools that rely on interest values, such as guidance, information hiding, or labelling to support analysts in interpreting complex results.

ACKNOWLEDGEMENTS

This work has been funded in part by the Innovation Fund Denmark through the Grand Solution project *Hospital@Night*.

8 STEERING-BY-EXAMPLE FOR PROGRESSIVE VISUAL ANALYTICS

Marius Hogräfer, Aarhus University, Denmark
Marco Angelini, Sapienza University Rome, Italy
Giuseppe Santucci, Sapienza University Rome, Italy
Hans-Jörg Schulz, Aarhus University, Denmark

ACM Transactions on Intelligent Systems and Technology (TIST), Volume 13, Issue 6, 2022, pp.96:1–96:26 doi:10.1145/3531229.

ABSTRACT

Progressive visual analytics allows users to interact with early, partial results of long-running computations on large datasets. In this context, computational steering is often brought up as a means to prioritize the progressive computation. This is meant to focus computational resources on data subspaces of interest, so as to ensure their computation is completed before all others. Yet, current approaches to select a region of the view space and then to prioritize its corresponding data subspace either require a 1-to-1 mapping between view and data space, or they need to establish and maintain computationally costly index structures to trace complex mappings between view and data space. We present steering-by-example, a novel interactive steering approach for progressive visual analytics, which allows prioritizing data subspaces for the progression by generating a relaxed query from a set of selected data items. Our approach works independently of the particular visualization technique and without additional index structures. First benchmark results show that steering-by-example considerably improves Precision and Recall for prioritizing unprocessed data for a selected view region, clearly outperforming random uniform sampling.

8.1 INTRODUCTION

Progressive visual analytics (PVA) is a way to bring the user into the loop of long-running computations by visualizing intermediate results well before the final result is available [6]. This is particularly helpful when the dataset under analysis is either very large, the computation run over that data is very complex, or even worse when both are true; in short, whenever running the entire analysis would take too long. PVA allows not only monitoring the running computation, but also canceling it ahead of time once a good-enough result is shown [78], which user evaluations have shown to significantly outperform using "blocking", non-progressive systems in terms of insights gathered [127]. One of the most promising uses of PVA is for computational steering where the intermediate results are used by an analyst to identify subspaces of interest on which to focus the computational resources so as to prioritize their computation [82, 107].

From early on when computational steering was proposed, it was inherently tied to direct manipulation [70, 119]. Yet when wanting to use direct manipulation to steer a running computation in a PVA scenario, one quickly discovers a problem: Say, for example, we want to steer the computation by brushing a region in the still unfinished visualization of all computation results. Brushing that region would then mean to prioritize the data items inside, so as to focus computational resources on that region and to complete the processing and rendering of the data items inside before all other parts of the visualization. Yet in order to give them this preferential treatment, we already would need to know which data items will in the end be mapped into that region – i.e., we would need the visualization already to be completed to determine those data items.

Existing computational steering approaches for PVA deal with this conundrum either by circumventing it using a 1-to-1 mapping between data space and view space, or they maintain a spatial index structure over the data to perform such a reverse-lookup from view space region to data subspace. A 1-to-1 mapping is used by the SHERPA system [27] that limits itself to chart types, which employ data attributes as axes – e.g., scatterplots and line charts. Whereas the spatial index is used for progressive multidimensional scaling [118]. The index structure provides a binning of data items – those that are already rendered are binned based on their position in view space and those not yet rendered are binned based on their distance to the already rendered items in data space. Every now and then, a rebinning occurs that (1) updates the bins of newly rendered data items based on their now available position in view space, (2) subdivides overcrowded bins into smaller ones, and (3) recomputes the binning of the remaining unrendered data items based on these changes. As the bins are defined in view space, they can be overlaid as a grid-like structure on the projection and the user can select individual bins to prioritize their associated, unrendered data items in the progression. Both of these approaches pose considerable limits: The 1-to-1 mapping greatly reduces the visualization possibilities to only those visual representations that use data dimensions as visual dimensions. Whereas spatial indexing requires periodic updates to keep current with respect to any newly rendered data. As each of these updates incurs a full pass over the data, this limits the applicability of this approach to only moderately sized datasets.

In this paper, we propose a third approach to this problem that can be utilized for progressive computations over multivariate, numerical data whenever other steering approaches fail – i.e., when no 1-to-1 mapping between view and data space exists and when the dataset is too large for periodic updates. This third approach relies on the idea of "query by example", where we use the data items already inside a selected region to find those that are similar and thus likely to be drawn in the selected region as well. To that end, we make use of decision tree classifiers, which have already proven useful for estimating user interest in data subspaces in non-progressive scenarios [29, 47]. The main idea is to train a decision tree that discerns between those data items already rendered inside the selected region and those outside of it. We then use the tree's decision rules to form SQL queries that return more, unrendered items of the "inside" case. To this end, we make the following contributions to the field of PVA:

- We introduce steering-by-example as an application of query-by-example for prioritizing data subspaces of interest based on selections in view space during incremental computations.
- We present a quantitative validation of steering-by-example in a series of benchmarks, showing that it significantly outperforms random uniform sampling in retrieving data for a given selected region in view space.
- We present ProSteer, an experimental visual environment for exploring steering-by-example, which is available as open-source.

8.2 RELATED WORK

PVA divides long-running computations into small steps. As a result, analysts using PVA can interact with this ongoing process and adjust it while it is still running, facilitating progressive data exploration. According to Mühlbacher et al. [82], such interactions with an ongoing computation can be distinguished into two groups: *result control* (what the computation does) and *execution control* (how the computation does it).

Result control is defined as any "interaction with the ongoing computation in order to steer the final result" [82]. This encompasses, for example, the early validation of a result being computed and a potential reparametrization of the computation if the result does not meet the analyst's expectations. One type of result control is the *inner result control*, which is based on partial results being generated by an ongoing computation that can then be adjusted on the fly. The existing literature in PVA has particularly looked at the implications for UI design incurred by adjusting the parameters of a computation while it is running [10]. Complementary to this type of result control, there can also be *outer result control*, which is based on final results generated by multiple computations. Instances of this type are TPFlow [71] and the work by Xie et al. [121], which gradually lead analysts from a computation that produces an overview visualization towards computations that bring out increasingly more detailed patterns in data subspaces.

Execution control is defined as "any kind of control of the execution of the ongoing computation of the process as such" [82]. The first type of execution control and often cited benefit of PVA is the ability to cancel the computation early on, once a good-enough result has been obtained. Early cancellation has been an integral part of PVA from its inception [107] and has subsequently been shown to increase analysts' efficiency in the analysis [127]. Its main challenge is the potential to cancel a computation too early while larger changes to the result are still inbound. PVA research has thus looked at approaches to recover from such situations [59, 80]. A second type of execution control is prioritization, which refers to adjusting the order in which data is processed by a progressive computation. The idea is that data of higher interest to the user should be processed before data of lesser interest, so that the generated partial results will reflect data of interest as early as possible. This form of control is also called *interactive steering*. State-of-the-art steering approaches include SHERPA [27] and MDSteer [118].

Outside the field of PVA, there exist some approaches to facilitate prioritization. ForeCache [12] and IncVisage [97] are examples of interactive approaches for prioritizing data, while explore-by-example [29] is used to steer queries in a progressive context. In particular the latter is of interest because of its general applicability: It extracts decision tree rules from a set of exemplars denoted by the user as being of interest. It then uses these rules to prioritize similar data elements.

Putting our approach of steering-by-example in this context, it is a method for execution control – specifically for prioritization of relevant data in the processing order. To do so in a generic way that does not depend on the type of visual mapping (as SHERPA [27] does) or on the ability to compute and maintain a spatial index (as MDSteer [118] does), our steering-by-example approach follows the idea put forth in the explore-by-example approach [29] and employs decision trees for prioritizing data items of interest.

8.3 THE STEERING-BY-EXAMPLE APPROACH

Next, we present steering-by-example as an approach for prioritizing subspaces of data in progressive visualization based on selections in view space. The following sections first introduce the scenario that steering-by-example addresses and then outline the algorithmic steps of our approach. Finally, some practical extensions to the general approach are discussed, which become relevant when implementing steering-by-example in a PVA system.

8.3.1 *The Steering-by-Example Scenario*

In the following, we denote the situations that benefit from our approach. To that end, we assume a function f that transforms a dataset D into D' with $D' \subset \mathbb{R}^2$. As we are specifically proposing steering-by-example for PVA, the computation of f should be complex enough to warrant the use of progression for a dataset of size $|D|$. Apart from this basic setup, steering-by-example makes no further assumptions about f. This makes steering-by-example a good fit for scenarios in which the details of f are either unknown due to the use of closed source software, or unspecified as would be the case when implementing a generic PVA library that is to work with any conceivable user-defined visualization technique. In its generality, the steering-by-example scenario specifically includes the following three cases:

1. The dataset size $|D|$ is too large to iterate over D multiple times, effectively prohibiting its binning and re-binning into a spatial index, as it is done for the steerable, progressive MDS [118].
2. The function $f^{-1} : D' \to D$ is unknown or does not exist at all (for instance when f's bijective property cannot be guaranteed due to dimensionality reduction), effectively prohibiting the direct lookup of all data items in a selected region of interest of D', as it is done for 1-to-1 mappings [27].
3. The function f is governed by a set of changing query parameters $\{p_1, \ldots, p_k\}$ – e.g., user location or date and time – that makes it impossible to precompute f for a wide range of possible data values from D and store the results for a table-based, reverse look-up of $D' \to D$.

Steering-by-example is able to handle these scenarios by (1) touching each data item at most once, (2) being agnostic about the used visualization, and (3) being computationally inexpensive enough to be used for highly context-dependent, ad-hoc analysis scenarios. Or even more succinctly: When all existing methods fail, steering-by-example is still applicable.

8.3.2 *Description of the Approach*

Next, we describe the steering-by-example method along its four phases. Each phase corresponds to a distinct state of the underlying decision tree classifier that drives our approach. We use a decision tree as underlying classification model, which work by Dimitriadou et al. [29]

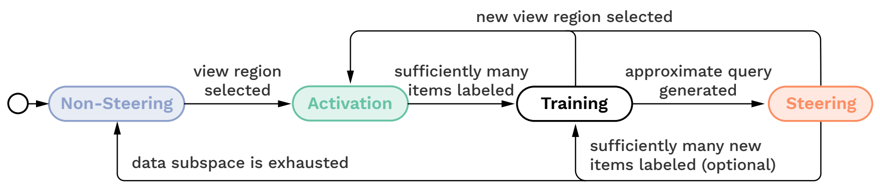

new view region selected
view region selected
sufficiently many items labeled
approximate query generated
Non-Steering
Activation
Training
Steering
data subspace is exhausted
sufficiently many new items labeled (optional)

Figure 34: State diagram depicting the four phases of steering-by-example and the transitions between them.

has shown to perform well for data exploration, due to the following properties: (1) It can be trained quickly for interactive use, (2) it produces sufficiently powerful classification models from relatively small inputs, allowing its use early on during the progression, and most importantly, (3) its model can be easily translated into SQL rules. Other classification techniques like SVM or deep neural networks do not fulfill these requirements, in particular the latter. A state diagram of the approach is depicted in Figure 34. The four phases are:

1. The *non-steering* phase is the default phase in which the user has either not yet selected a view region of interest or no further data items in the dataset D match the last steering query.

2. The *activation* phase is when data items are labeled for training based on whether they are located inside or outside a selected region of interest.

3. The *training* phase is when the decision tree classifier is trained based on the labeled data items.

4. The *steering* phase, in which a query approximating the characteristics of data items inside the selected view region is constructed from the decision tree rules and used to retrieve more data items likely to fall into that region.

We describe these phases in more detail below, explaining the main goal of each phase and the conditions for transitioning between them.

1) NON-STEERING PHASE: Initially, the PVA system does not have any indication by the user what data is of interest, as no selection has been made in the view space.

The goal of this initial phase is thus to give the users a first look at the data, allowing them to identify interesting regions that should be prioritized. Since the user interest is open at this point, this phase of steering-by-example uses a default sampling method of the dataset D based on the query parameters $\{p_1, \ldots, p_k\}$ in order to retrieve the next chunk of data, and the retrieved data items are not yet labeled as relevant or not. In our case, this default sampling method is random uniform sampling. In addition to being the first phase a PVA system runs after launch, the non-steering phase also functions as fallback for the steering phase, once all data items that match the approximate steering query have been retrieved. Then, the system will continue to retrieve further data items from D using default sampling. This "falling back" essentially resets the steering-by-example algorithm.

2) ACTIVATION PHASE: Once the user selects a region of interest, the system enters the activation phase. This selection can be made through any interaction in view space that identifies a region that the user is interested in. In our case, we use brushing directly on a two-dimensional rendering pane, notwithstanding that the approach could also work for other selection methods nor for higher dimensions, such as three-dimensional volumetric renderings provided a suitable brushing mechanism [66, 125]. As the user can perform the selection at any time during the progression, this phase can be reached from any other phase in the steering-by-example algorithm.

The goal of this phase is to gather a sufficient number of data the user is interested in, to guarantee an adequate quality of the classification model for steering later on. All newly retrieved data items are thus labeled as relevant or not, based on whether they lie inside or outside the indicated region of interest. In cases where data from previous iterations is still available, this data can also be considered for labeling, yet the steering-by-example algorithm does not require any caching. The system remains in this phase until sufficiently many data items are labeled as relevant. The training of the classification model has not yet been started, thus the system continues to use default sampling to retrieve the next chunk of data while in the activation phase.

3) TRAINING PHASE: Once enough data items have been labeled as relevant, the training phase begins.

The goal of this phase is to produce a query representing the approximate inverse mapping from the selected data items of interest to data properties they, and only they have in common. These properties are then to be used to query for more data having the same properties and prioritizing their computation – thus effectively steering the progression towards other relevant data. As these properties are still being established, the system continues to use its current query while training. All data items newly retrieved during this phase continue to be labeled as relevant or not. The system remains in this phase until the training of the classification model is completed. In practice, the training phase lasts only for a short period of time, depending on the complexity and size of the training dataset, due to the low computational complexity of training the decision trees.

Note that the system can also enter this phase from the steering phase, if enough new data items are located inside the selection to trigger a refinement of the classification model based on that newly available data. Re-entering the training phase is however optional, as rebuilding the model often will lead to only marginal improvements of the steering quality once large portions of the dataset are already computed.

4) STEERING PHASE: Once the used classification model is constructed, the system enters the steering phase.

The goal in this phase is to provide the user with all relevant data items from the dataset D that match the approximate query, by retrieving data items that will be likely plotted close to those in the view selection. In contrast to all previous phases, the system thus uses the steering query extracted from the decision tree to retrieve the next chunk of data, thus steering the progression towards interesting data subspaces. For decision trees, an SQL query for steering is constructed from the classification model as follows: Each path from the root to a leaf node in the decision tree model represents a set of decision rules that must be satisfied in order for the decision tree to classify it as relevant. Thus, the generated steering query is equal to the logical disjunction of conjunctions of these sets of rules. During the steering phase, all retrieved data items continue to be labeled as relevant.

The system exits the steering phase for two reasons: The first reason is that the data subspace indicated by the approximate query is exhausted, in which case it falls back into the non-steering phase. The second reason is that enough new data items from the steering query fell inside the region of interest to trigger further refinement of the query, in which case the system returns to the training phase.

8.3.3 *Extension to the Basic Approach*

The basic approach outlined above assumes that the user selection in view space (at some point in time) contains sufficiently many data items to train the decision tree classifier. Yet, this is not necessarily always the case. Specifically, there are two cases in which not enough relevant items can be collected: Either, the region is temporarily too sparse, but sufficiently many data items of the data will land inside the selection in the future, or the region will always be too sparse, even when waiting until the progression completes.

One strategy for addressing this challenge is that the threshold for what counts as "sufficient" could be a user-definable hyperparameter to the algorithm. Then, experienced users could lower this value based on their particular use case and thus increase the chance of engaging the steering. However, this is always a trade-off with the quality of the classification model, which usually benefits from having a larger training dataset.

An additional, automated strategy is to artificially increase the size of the user selection each time no data item from the newest chunk of data falls into the selection, thus also considering data outside the original selection for training. Intuitively, this approach is intended to "broaden" the range of data that is considered interesting, thereby increasing the chance that sufficiently many data items can be collected. Conceptually, this approach resembles a query relaxation of the steering query [81]. The motivation here is that data inside this "broadened" selection remains at least somewhat interesting to the user, as it is rendered close to the region of interest. However, this approach will generally lead to a worse Precision during the steering phase compared to a model trained without increasing the selection, but nevertheless remain more beneficial to the user's analysis than random uniform sampling. The degree to which the extent of the selection is increased is another hyperparameter to steering-by-example, for instance using a percentage-based increase in size. One could also make the growth proportional to the number of chunks that contained no data items located inside the selection. Then, the chance of "getting a hit" could increase with every chunk without a data item inside the selection, as the selection grows at a greater rate.

Another extension for this is to wait for a certain number of iterations before the size is increased, instead of increasing the size with every "fruitless" iteration. The idea here is to avoid reducing the steering quality for selections that only as an artifact of sampling remain empty for a single iteration, but generally are densely populated. The number of chunks before the box increases would be an additional hyperparameter for steering-by-example.

8.4 BENCHMARKS

This section reports the results about the performance of steering-by-example for progressive visual analytics. In order to test the proposed solution, we defined a set of automatic test cases on which we collected evaluation metrics. We first describe the overall obtained results, and then we detail the testing environment and the measures we collected with it.

8.4.1 *Test Results*

We evaluated steering-by-example on a sample of the AirBnB dataset for Paris which consists of $64,216$ data items for listings of housing options, with each listing being described by 47 dimensions, containing both numerical and categorical values. The dataset was obtained from InsideAirBnB [1]. While this dataset is relatively small, the additional computations we run on each chunk make the incremental use case worthwhile, as a full pass over the entire dataset could take up to ten minutes.

We compared steering-by-example with random uniform sampling, in which the progression is not steered, and the data space is uniformly sampled, serving as the average case. While more sophisticated sampling algorithms exist [126], we chose random uniform sampling as our baseline, as it still remains the de facto standard sampling approach in PVA literature, since it is widely available across programming languages and frameworks, and performs reasonably well on large datasets. We tested the two approaches on more than 1000 test cases, evaluating the performances with respect to Average-Precision and Recall metrics.

We measured a clear advantage of the steering-by-example approach over random uniform sampling, both in terms of average Precision (ca. 10 times higher) and Recall (ca. 4 times higher). Additionally, our results show independence of performance from chunk size and identify a good threshold for starting the activation phase and obtaining good performances in just 20 items. We show that the steering-by-example approach converges to the expected results much faster than random uniform sampling, similar to the hypothetical, perfect performance of direct lookup, with only negligible overhead with respect to both per-chunk computation time and overall time. Finally, our benchmarks show how steering-by-example scales even to large datasets.

Figure 35 reports a summary of the whole experiment, showing the Average Precision and Recall box-plots for steering-by-example and random uniform sampling, clearly showing how steering-by-example outperforms random uniform sampling in both metrics.

8.4.2 *Test Cases*

In order to test the effectiveness of the steering-by-example solution in a systematic way, we instrumented a fully automated benchmark with the goal of producing a large and significant set of test cases. We use the following mapping function f for any given listing x in the AirBnB data: $f(x) = (priceSavings(x), walkingDistance(x))$. This function thus produces a tuple, containing the price difference to other listings and the walking distance to a fixed location of interest in the city. To reduce the precomputation overhead of the first part, we limit the benchmarks to a subset of the AirBnB data to include only listings within in a 60 to 90 Euros price range, limiting the total query size to $25,922$ items.

1 http://insideairbnb.com/get-the-data.html

We evaluated steering-by-example against two baseline conditions. Random uniform sampling forms the **lower baseline** condition, which retrieves the data in random order, regardless of the selection. As **upper baseline** we use the state-of-the-art approach of direct lookup that is also implemented by SHERPA [27] and that will behave like a perfect predictor: it always exclusively retrieves items inside the selection, until all data for the selection is processed. It should be noted that, because direct lookup requires a 1-to-1 mapping between data and view space which is not available for the mapping function we use in our scenario, we have to precompute all values needed to build a 1-to-1 mapping w.r.t. the user selections used in the test (latitude, longitude, min price, max price, and range of walking distance among alternative hotels). The time for precomputing this data for a single user test is about 15 minutes (on a quad core i7 and using indexed relational tables in MySQL). Obviously, it is not possible to precompute *all* data for *all* possible user selections of the five parameters of latitude, longitude, min price, max price, and walking distance. Even partitioning the user's selection domains in discrete intervals (e.g., selecting latitude and longitude in steps corresponding to 500 meters, the price range in intervals of 5 US$ between 60 and 100, and the walking distance in a range of 100 meters between 100 and 500) makes the computation time not feasible (about 70 months if limited to a 15×15 km square region in Paris). Moreover, the results of these precomputations must then be stored in a lookup table that allows to directly retrieve the data items inside a selected screen region. The size of this additional data is about $225,000$ attributes per AirBnB listing.

We considered as parameters of this benchmark:

- *Chunk size*: the chunk size of the progressive process on three levels (50, 100, and 150 items per iteration)
- *Activation threshold*: the minimum number of items that must be inside the selected view region to trigger the training phase of steering-by-example (10, 20, 40, 60, 80, and 100 items each)
- *Query result cardinality*. We split the test cases into three groups based on the cardinality of each selected view region with respect to the full query:
  - *Low cardinality*: this set is formed by selected view regions containing a number of items ranging from 1% to 4.5% of the full query
  - *Medium cardinality*: this set is formed by selected view regions containing a number of items ranging from 4.5% to 22.5% of the full query

- *High cardinality*: this set is formed by selected view regions containing a number of items ranging from 22.5% to 50% of the full query

The rationale behind this choice is to characterize the performance of steering-by-example for different cardinalities. More in detail, we do not go over 50% of the query cardinality because at that point the probability to correctly identify a point as part of the actually selected view region or not is equal (or higher) than chance, and results' validity would be affected. For this reason, even if the theoretical upper limit is 50%, the randomly generated selected view regions have a maximum cardinality of $\sim$ 40%. We randomly generated 20 view selections per group without repetition. This results in 60 selected view regions tested, that cover in a good way the variability of the region size and position on the screen (details can be found in the supplemental materials).

The combination of these benchmark parameters (3 chunk sizes, 6 thresholds) led to 18 runs per selected view region, and given the 60 query cardinalities resulted in $1,080$ runs for steering-by-example and $1,080$ for random uniform sampling. The data needed for the direct lookup approach was precomputed one time for each <*selected view region, chunk size*> combination, given its independence from any other parameter. We let each run execute until 100% Recall was obtained. These test cases included selected view regions of different sizes, aspect ratios, and density of the contained points in order to capture many different scenarios.

8.4.3 *Detailed Measures*

We tested the three approaches on all the test cases, evaluating the performances with respect to Precision, Average Precision, and Recall metrics. These metrics were computed from the standard binary classification variables with regards to whether an approach classifies or misclassifies a data item as inside or outside the selected view region.

Figure 35 reports the high-level comparison between steering-by-example and random uniform sampling for Average Precision (Precision computed on average for a single run) and Recall: specifically, we collected those data at the iteration in which steering-by-example ends its effects (when the system switches from Steering phase to Non-Steering phase), which can vary depending on each run and on the cardinality of the tested query. For example, for the medium cardinality group, we collected the statistical median at iteration 32. For steering-by-example, the median value of the Recall is at 0.84 and the median value of Average Precision is at 0.77, both much higher than the respective values for random uniform sampling (Avg-Precision

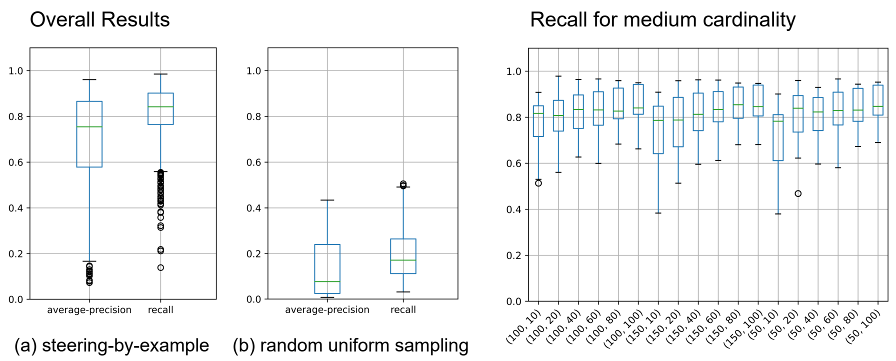


Figure 35: Left: Average Precision and Recall comparison for steering-by-example and random uniform sampling on the full set of test cases. The figure shows that steering-by-example clearly outperforms random uniform sampling for both metrics. Right: Recall results for medium cardinality group split by chunk size (100, 150, and 50 items respectively) and activation threshold (10, 20, 40, 60, 80, and 100 items respectively). The figure shows that chunk size has little to no effect on the Recall behavior. Conversely, activation threshold shows that from 20 items value the results tend to saturate, with only 10 values showing a consistent decrease of performance due to higher variability. Precision and Recall of the direct lookup approach are omitted, as these are – given that all the data for the 1-to-1 mapping is precomputed – consistently at the maximum values, performing better than steering-by-example.

median=0.08, Recall median=0.18). While the Recall box for steering-by-example is very compact, showing that the effects of steering-by-example are valid for the majority of the test cases, the Average Precision box is more spread. This effect can be explained by runs in which the cardinality of items included in the selected view region is lower (specifically for the low cardinality group). Additional charts showing the performances split by query cardinality groups are present in the supplemental material. Even looking at the split group performances confirms that the steering-by-example solution achieves results much better than the random uniform sampling. Finally, the outliers present for the steering-by-example Recall are all relative to configurations in which the query result cardinality is very low (near 300 items) and the activation threshold is at the minimum (10).

Nonetheless, even for those "more difficult cases", steering-by-example obtains better performances than random uniform sampling in the same extent of the other cases, even if with overall lower values for Recall. For those reasons, we claim that, independently of the query cardinality, steering-by-example obtains much better results than random uniform sampling for both Average Precision and Recall.

We then inspected the obtained results with respect to the chunk size and activation threshold. Figure 35 shows Recall values for the medium cardinality group (the other groups' performances are re-

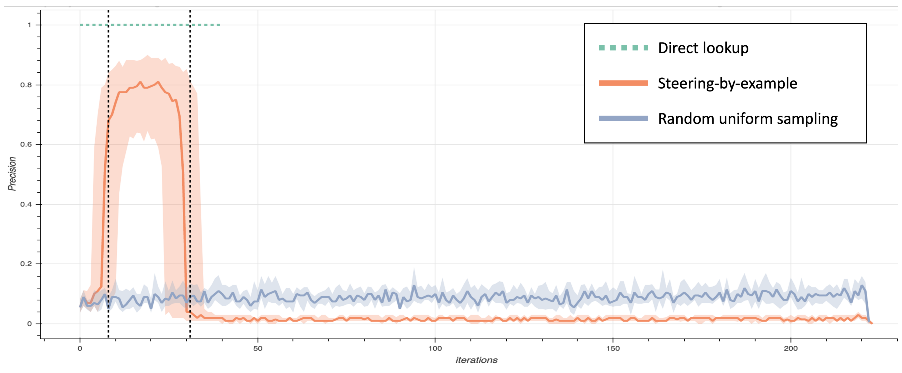


Figure 36: Comparison of Precision trends for the direct lookup, steering-by-example, and the random uniform sampling cases. Comparison of median values shows that steering-by-example outperforms random uniform sampling for all the iterations belonging to the steering phase, with median at 0.8 less than 20% distant from the perfect predictor. In the Non-steering phase, the Precision values are very similar, with random uniform sampling slightly better due to a higher number of items left (resulting in a higher probability of finding the remaining data items). Precision of the direct lookup approach is consistently at 1, provided that the necessary 1-to-1 mapping has been precomputed.

ported in the supplemental material). We can observe that results do not show a significant effect of chunk size: in fact, the box-plots show a similar trend with respect to the three chunk sizes with which we experimented. This led us to discard this parameter for further analysis and consider it set up by default at 100 items. The chunk size only affects the speed of the progressive process and not its quality. On the other hand, the activation threshold shows a slight effect on both the median values and the compactness of the resulting box-plot, with higher thresholds yielding slightly more compact plots. Median values are meanwhile less affected, ranging (similarly) between values of 0.78 to 0.84, confirming what evidenced for the general case. On the lower end of the results, only test cases with an activation threshold set to 10 show degradation of results, with a minimum slightly below 0.4 for Recall. Even if those worst cases are still comparable to random uniform sampling best cases, we suggest setting up the steering-by-example with an activation threshold greater than 10 for maximizing the performances.

Having discussed all the parameters, we move on to comment on the temporal trends, considering the medium cardinality group and chunk size set at 100 items. For all the approaches we report for all the experiments the trends of Precision and Recall metrics per iteration. Figure 36 shows statistically aggregated results on the Precision metric. The median values trend for each curve is reported with full color hue, while the alpha blended areas identify the variations between the upper and lower quartile values. In this way the statis-

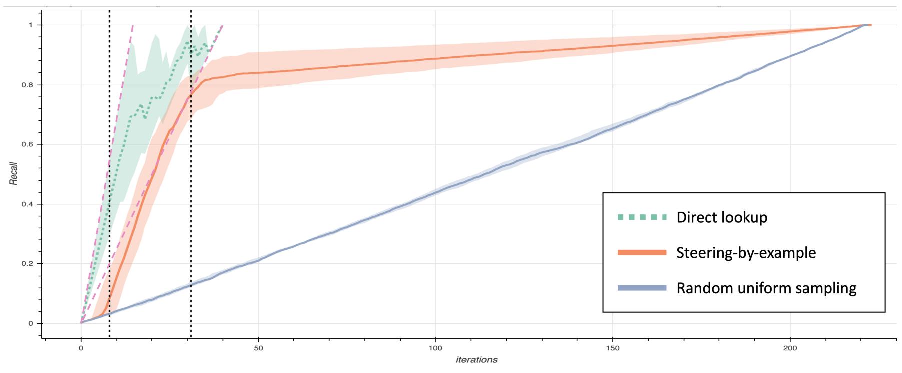


Figure 37: Comparison of Recall trends for the direct lookup, steering-by-example, and the random uniform sampling cases. The figure shows how the steering-by-example Recall trends rise with similar speed to direct lookup during the steering-phase, and clearly outperforms the random uniform sampling with $\mathit{median} = 0.77$ for steering-by-example and $\mathit{median} = 0.12$ for random uniform sampling at the median iteration in which the steering-phases end (iteration 32). The Recall of direct lookup (delimited by magenta dashed lines for reference) is consistently better than steering-by-example, given that a suitable 1-to-1 mapping has been precomputed.

tical variability is reported for all curves. Two vertical black dashed lines report respectively the median value of the starting iteration and the median value for the ending iteration for steering-by-example: those lines identify the statistical extension of the steering phase for steering-by-example.

We can observe how Precision values are consistently high in the Steering phase, with median values just lower than 20% with respect to the flat line representing direct lookup. Additionally, it shows how steering-by-example is consistently better than random uniform sampling. In the Non-steering phase, the performance of steering-by-example becomes worse than random uniform sampling, even if at that moment both techniques are the same (random uniform sampling). We explain this behavior due to a lower residual probability of finding the (few) items left in the steering-by-example case.

The good performances of steering-by-example are confirmed by evaluating the Recall trend reported in Figure 37 for direct lookup. It is interesting to note how the steering-by-example Recall trend rises similarly to the perfect predictions of direct lookup, reaching very fast (a little more than 20 iterations, eventually lower if the chunk size is raised) the 0.8 level of Recall. After that point, the remaining 0.2 are achieved by reactivating random uniform sampling, which creates the long tail that at some point (slightly before the random uniform sampling) converges to the Recall 1.0. This slow convergence towards full Recall after the steering phase could be sped-up by executing a new training and steering phases immediately after the end of the previous steering phase.

Overall the benchmark demonstrated how steering-by-example outperforms the random uniform sampling with respect to Precision, Recall, and speed for all the tested cases. Full Benchmark results are available in the supplemental materials.

8.4.4 *Implementation Details*

Here we briefly outline our implementation underlying the benchmarks. More details, as well as source code of all components including the code used for benchmarking is publicly available as open-source on GitHub[2].

Our implementation of steering-by-example is written in Python 3.8 around the scikit-learn machine learning library [92] for its implementation of CART decision trees [15, ch. 2.3], as well as the Pandas and numpy libraries [112]. The datasets we use in our benchmarks are stored and accessed from a MySQL 8[3]

8.4.5 *Threats to Validity of Benchmarks*

Our benchmarks demonstrate the applicability of the steering-by-example approach under certain assumptions. Here, we want to explicitly state the particularities of our evaluation that need to be taken into account when interpreting the results.

A first consideration is the **data type** used in the benchmarks. We have evaluated steering-by-example on a dataset containing numerical dimensions. While decision trees can generally also be trained on categorical data, we cannot make any conclusions about their performance for this data type.

Another consideration is that our evaluation relied on **rectangular selections** in view space. This design decision was made for consistency between benchmarks and to reduce the controlled variables in our testing, ensuring that all benchmarks use regularly-shaped selection boundaries. Generally, steering-by-example does not rely on any particular selection mechanism. All the decision tree requires for training is a set of data items labeled as relevant. In a series of informal primary tests using the lasso selection tool in ProSteer, we also observed that our implementation can handle more complex selection shapes. Nevertheless, our benchmarks cannot guarantee that the performance of steering-by-example also applies to more complex selection boundary shapes.

2 `https://vis-au.github.io/prosteer`

3 `https://www.mysql.com` database. All tests were run on an Intel Core i7 processor, running at 2.7 GHz, with 16 GByte of RAM. The data was stored on a 1 TByte SSD.

items	training	rules	recall
100	2.1ms	73ms	0.85
150	2.5ms	110ms	0.82
200	3.3ms	163ms	0.85
250	4.3ms	224ms	0.84
300	4.5ms	248ms	0.83
350	5ms	342ms	0.87
400	5.8ms	395ms	0.87
450	6ms	426ms	0.86
500	6ms	494ms	0.86

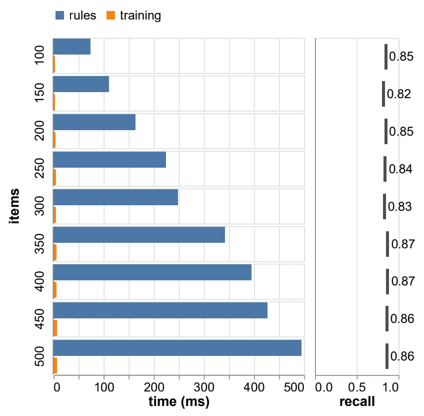


Figure 38: Testing the impact of the number of items used for training the decision tree classifier on average training time and average time to extract the conditional rules, as well as the average Recall of the generated query, based on $N = 20$ runs. The concrete values are shown on the left, and a visualization of that data on the right. The results show that, while the time needed to extract an SQL query increases linearly with the number of items, the Recall does not increase noticeably and remains at around 0.85.

A third consideration concerns the evaluation of **computation time** for the tested approaches. Under our settings, processing a single chunk requires 0.4 seconds to complete for all three approaches. The steering-by-example approach is the only one that requires additional time to train the decision tree classifier and extract from it the decision rules (training phase). To assess the impact on the overall retrieval time, we conducted a series of experiments in which we measured the impact of the number of items on the time it takes to train the decision tree classifier, the time it takes to extract the conditional rules in the training phase, and finally the Recall that the query produces. Figure 38 gives an overview of the results. Our observations are in line with previous work evaluating the impact of training dataset size on the performance of decision trees [85], in that the size of the training dataset did not benefit performance, while increasing the complexity of the tree structure. Overall, we conducted 20 runs for 9 different numbers of items, ranging from 100 to 500 in increments of 50. Average training times ranged from 2ms for the 100 items cases to 6ms for 500 items. This makes the training time not significant with respect to the overall time needed for both one iteration (400ms) and the overall process length (220 iterations $\times$ 0.4sec $=$ 1 minute and 28 seconds). Regarding the rule extraction time, we report 73ms for 100 items. Again, this time represents 18.25% of the iteration time, as it is "paid" only once when the steering phase is activated, and 0.82% of the overall time. Therefore, the cost introduced by training the decision tree is negligible in the scope of a full computation, with a progression using steering-by-example terminating only imperceptibly later than a progression using random uniform sampling. Overall, rather than being limited by long training times from selections on very dense data, these findings suggest that we can keep training and extraction times of our approach consistently low, by drawing a fixed-size sample from those selections, and still provide a good steering performance.

For reproducibility, we removed any dependency from computing and network communication performance by precomputing the walking distances from the actual user-selected location in Paris to the relevant listings using Euclidean distance and simulating a call to the Google API introducing a delay of 0.04 seconds for each call, as well as the saving opportunity for each, by computing the difference between its price with listings within a radius of 300 meters. The respective values were stored as the *Distance* and *Savings opportunity* attributes for each listing in the database. As the main design goal for our interactive environment was primarily benchmarking our approach, it thus does not allow producing listings for any arbitrary place on a map. Yet, computing the relevant listings for a single location on-demand can take hours, which was therefore not feasible to evaluate our method without precomputing the two measures. In addition to saving time, precomputation also isolates any potential fluctuations due to differing computation times from the results.

8.5 PROSTEER: AN EXPERIMENTAL VISUAL ENVIRONMENT FOR STEERING-BY-EXAMPLE

Here we present ProSteer, our interactive visual demonstrator for steering-by-example, which can be used to experiment and test-drive our approach. ProSteer is a client module to steering-by-example, in which the data is visualized and regions of interest can be defined in view space. ProSteer is not intended as a stand-alone, general-purpose visual analytics tool, but is instead designed for demonstrating steering-by-example. Thus, while the automated, "command-line" benchmarks reported in the next section provide a numerical perspective for assessing the performance of the approach, ProSteer can be used to illuminate the user perspective of utilizing steering-by-example in visual analytics.

More specifically, ProSteer is designed to support the following tasks: (1) Compare the progression using steering-by-example with a progression using random uniform sampling, (2) make selections in view space, (3) explore the progression at one point in time, (4) compare the progression at different phases of the algorithm, (5) compare data inside the selection with the remaining dataset. ProSteer is implemented using the D3 visualization library [14], together with the React[4] and TypeScript frameworks. In the following, we will describe

4 https://reactjs.org

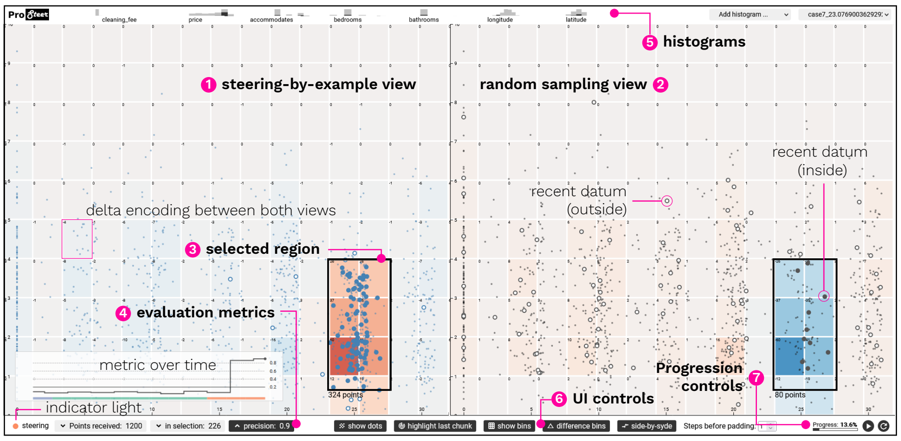


Figure 39: Screenshot of ProSteer.

the interface of ProSteer along these requirements, using the labels ① - ⑦ from the screenshot in Figure 39 to refer to its visual components.

The center of the interface is split in half, showing the progression on the same dataset in two views: The steering-by-example view ① shows the progression using our steering mechanism, while the random sampling view ② shows the baseline case. A heatmap in the background encodes the delta in the number of items that were retrieved in a grid cell of the view, and individual data points are rendered on top of that, with recent points rendered larger to highlight new data inside and outside the selected region ③. Evaluation metrics ④ show how steering-by-example performs over time in a line chart based on the current phase, and histograms ⑤ show the distribution of data inside vs. outside the selection. Interface ⑥ and progression ⑦ can be controlled through widgets at the bottom.

8.5.1 *Comparing steered with non-steered progressions*

The first question we want to answer in ProSteer is whether and by how much the user gets to see interesting data faster using steering-by-example. ProSteer addresses this question through its central view that takes up most of the screen space. Data items retrieved by the steered progression (①) are visualized in a side-by-side view with a non-steered progression (②) over the same dataset. Through this juxtaposition, one can compare how steering-by-example affects the overall distribution of individual data items across regions of the view space. Other visual encodings of ProSteer rely on this side-by-side view to facilitate further comparisons.

For instance, as the changes in data layout become less apparent in later stages of the progression, when many data items are already plotted and new ones do not stick out as much, the latest chunk of items retrieved from the computation module is additionally highlighted as larger, fully opaque points. If a point is located inside the

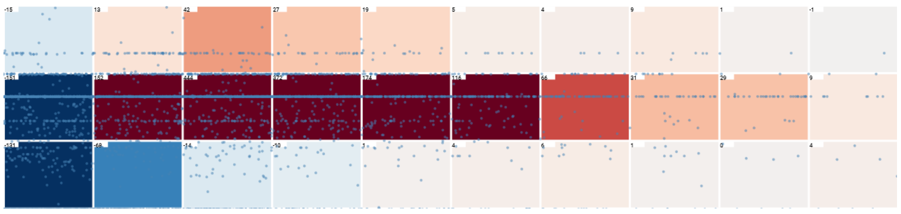


Figure 40: Enlarged screenshot of the delta encoding in ProSteer (see ① and ② in Figure 39): color values in a heatmap encode the difference in the number of items between the progression using steering-by-example and using random uniform sampling per grid cell. The darker the red tone, the more items lie in that cell in this view, and the darker blue, the less data in that cell. If the number of items is equal, grid cells have a neutral gray tone. The number in the top left corner shows the absolute delta value per cell. On top of each cell, individual data items are encoded as dots along axes of a scatterplot (X: price saving in the neighborhood, Y: walking distance to a point of interest on the map), which are computed from the data after they are retrieved from the database and thus cannot be directly used for building an SQL query.

selection, its fill color is either blue or black depending on the view, and it is white otherwise. This encoding allows to assess qualitatively, how the currently sampled region of the view space differs between the two progressions. In both views, the selection in view space of a region of interest is shown, indicating the number of data items contained in each. Additionally showing the numeric value of data items supports the quantitative assessment of steering on the one progression compared to the unsteered progression.

In addition to showing individual data items, ProSteer renders a grid-based heatmap in the background of the scatterplots (see Figure 40). Each cell of the heatmap encodes the difference in the number of data items that are rendered inside the particular region of the view space compared to the same position in the respective other view. A diverging color scale is used that encodes negative values as blue, positive values as red, and the neutral point where both progressions are equal as grey color hues. This encoding helps to get a quick impression about how much a region in view space differs in the steered and non-steered progressions, for instance during the steering phase. In addition to a qualitative assessment of the differences based on color, each cell in the heatmap also shows the numeric value that it encodes in its top-left corner.

Implicitly, encoding the difference in this way also carries uncertainty information, i.e., it informs the user that the fact that a region currently appears to be dense in the steered visualization may indeed be an artifact of the steering, that could in later iterations "even out", as more data lands in other regions. Uncertainty about observed patterns in regions targeted by the steering is closely related to the general challenge of uncertainty caused by the progression that constitutes a research challenge in its own right [59]. Our focus lies on testing algorithmic aspects of the steering-by-example, so future work is necessary to evaluate whether our encoding is an effective uncertainty encoding.

8.5.2 *Make selections in view space*

Another central design goal for ProSteer is the ability for user-driven selections of regions of interest, to support an interactive, customized evaluation of steering-by-example beyond the pre-defined test cases.

To this end, the interface supports direct brushing on the view space (③). ProSteer allows defining both rectangular selections like those used in the benchmarks as well as custom shapes using a lasso tool. Alternatively, the exact selections used in the benchmarks can be recreated using the dropdown menu in the top right. Moreover, ProSteer implements an extension to the basic steering-by-example approach, in that the size of that selection is increased with every chunk for which no data item was located inside the selection. As described in Section 8.3.3, one can define a custom number of chunks that should be waited, before the size is increased, using the text box in the bottom row of the interface.

8.5.3 *Explore a progression at one point in time*

Another question for our benchmarks is how steering-by-example itself performs at a certain point in time. The ways in which ProSteer supports this task visualizing the state of the progression, visualizing evaluation metrics, providing widgets for customizing the UI, and controlling the progression.

The state of the progression is shown both in terms of the phase of the steering-by-example module and in terms of the progress of the computation. The phase for the latest chunk retrieved is shown as a small "indicator light" in the bottom-left corner of the interface. A progress bar (⑦) in the bottom-right corner of the interface in turn shows the percentage of data that has so far been processed by the progression. Next to that progress bar are widgets for temporarily pausing and resuming, and for fully resetting the progression.

To support exploration with steering-by-example, three evaluation metrics (④) are shown in small text boxes next to the indicator light: The number of retrieved items, the number of items that are located inside the view selection, and the resulting Precision of the latest chunk. These metrics give a quantitative view of the latest data chunk of the progression.

ProSteer also allows customizing the visual encoding for different analysis goals. Control widgets in the bottom row of the interface (⑥)

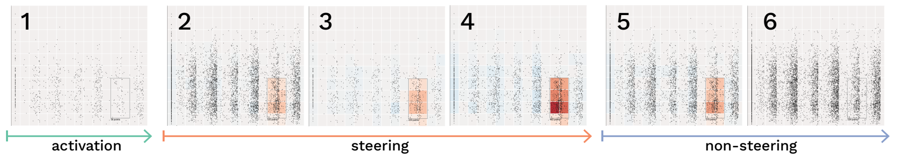


Figure 41: Screenshot series showing how the heatmap that encodes the difference in regional data density goes from cells in gray during the activation phase (1) to red cells around the selected region during the steering phase (2-4). When entering the non-steering phase, the heatmap equalizes to gray again (5-6).

for this purpose allow changing the encoding of the heatmap between absolute and delta values, toggling the side-by-side view on and off, and controlling how many data chunks without a data item from the selected region are permissible before the user selection is automatically grown in size. The top row allows setting the dimensions that are used for the visual encoding in the central view.

8.5.4 *Comparing a progression between different points in time*

Another question for our benchmarks is how the phases of steering-by-example affect performance over the duration of the progression.

For qualitative comparison, the locations of data items from the latest chunk are highlighted as opaque dots. In combination with the "indicator light", one can monitor how the distribution of the latest data items changes once the system goes in and out of the steering phase. Additionally, the heatmap encoding that visualizes the difference to a non-steered progression shows how during the steering phase, the sampling of the data gets skewed towards the selection in view space. As the progression continues, one can also observe how regional differences between the two progressions equalize (see Figure 41).

For quantitative comparisons, the system records the evaluation metrics over time as well as the phase of the algorithm that the system was in at each point. Line charts (④) that show the evolution of the metrics can be toggled when clicking on either of the text labels in the bottom left corner of the interface (see Figure 42). Below the line chart, colors indicate the phases of steering-by-example for each measurement. When hovering the mouse cursor over the line chart, individual values are shown in a tooltip.

8.5.5 *Comparing data inside the selection with the rest of the data*

The last question we can address with ProSteer is how data items inside the selection differ from the rest of the dataset.

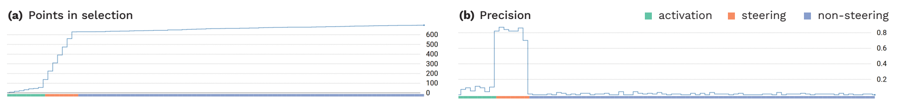


Figure 42: Enlarged screenshots from ProSteer's evaluation metrics (see ④ in Figure 39) visualizing the items inside the user selection (a) and the Precision of retrieved items being located inside that selection (b). The colored line below the line charts indicates the state of the steering module at a certain time.

To qualitatively answer this question, the top row of the interface contains histograms (⑤) that can be created for each dimension of the data. Each bar in a histogram shows the percentage of data items that lie inside the selection. Based on these histograms, one can compare whether the selection is equally distributed across a dimension, which indicates a representative sample, or whether the selection is skewed towards certain values (see Figure 43). The latter case is often beneficial to a better performance of steering-by-example, as it allows the decision tree to better separate the class of relevant data items from irrelevant data.

For quantitative comparisons, the view space selection shows the number of data items that are currently located inside it. In combination with the "Points received" metric, one can compare whether the sample contains large or small portions of the dataset.

8.6 USE CASES

In this section, we assess the generality of steering-by-example for two use cases, applying the approach to a different, larger dataset and on a different visualization using dimensionality reduction.

8.6.1 *Steering-by-Example on Large Datasets*

To assess scalability towards large datasets, i.e., considering the case in which the complexity of the f function comes from the dataset size |D| (see Section 3.1) and not by the computation, we have challenged ProSteer against the 112 million items dataset of the New York

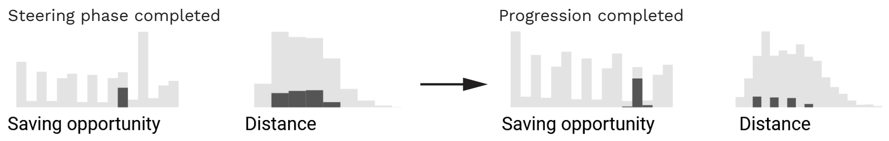


Figure 43: Enlarged screenshot of histograms in ProSteer (see ⑤ in Figure 39) showing the overall distribution of the data compared to the data in the selected view region. The left pair shows the histograms after the steering phase is completed and the right pair shows the data when the entire progression is completed.

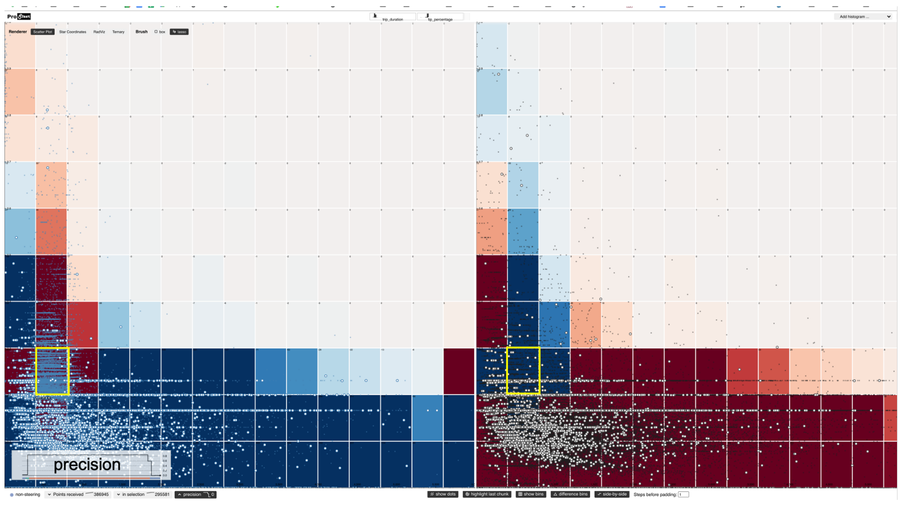


Figure 44: Screenshot from ProSteer, applying steering-by-example to the NYC taxis dataset.

City 2018 Yellow Taxi Trip [5]. As a mapping function for each ride x, we used $f(x) = (\mathrm{tripDuration}(x), \mathrm{tipRatio}(x))$, investigating the relationship between the duration of the trip (i.e., $\mathrm{tripDuration}(x) = x_{\mathrm{endTime}} - x_{\mathrm{startTime}}$) and the amount of the tip with respect to the fare (i.e., $\mathrm{tipRatio}(x) = x_{\mathrm{tipAmount}}/x_{\mathrm{totalAmount}}$). The size of the dataset prohibits us from performing a full benchmark on it: processing the whole dataset to compute the ground truth for a single use case requires 15 hours and replicating the full analysis done for AirBnB (about 1000 configurations explored) will require more than one year of computation. According to that, we just report on a qualitative test of the system, showing a comparison between steering and random sampling and terminating the analysis after the steering phase has been completed. Being the complexity of the plotting only associated with the size of the data, we have used a chunk size of $10,000$ whose elaboration lasts about 5 seconds; processing the whole dataset requires about 15 hours. The results are quite similar to those observed for the AirBnB dataset: the steering outperforms the random sampling with a significantly higher Precision. Figure 44 shows an example of one of the qualitative experiments we have performed with the system, after having plotted about $380,000$ trips.

The X axis shows the duration of the trip, the Y axis the ratio $\mathrm{tip}/\mathrm{fare}$. The steering phase handled about $350,000$ taxi trips, showing a consistent Precision of about 0.82; the Recall is not available due to the time needed to precompute the ground truth. At the end of the steering phase the steered approach (left side) shows about $280,000$ items in the user-selected area (yellow box) against the about $5,000$ of the random approach (right side). Moreover, the heat-map of the ran-

5 https://data.cityofnewyork.us/Transportation/2018-Yellow-Taxi-Trip-Data/t29m-gskq

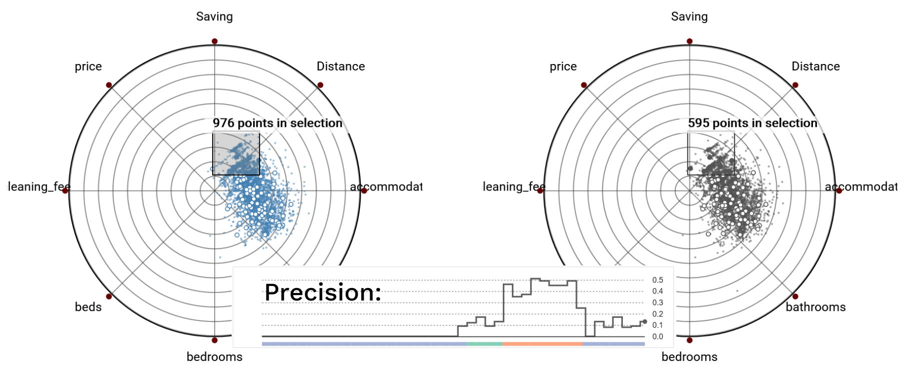


Figure 45: Screenshot from ProSteer, using steering-by-example with the RadViz dimensionality reduction technique, compared to the same progression that uses random uniform sampling for retrieval. The selected region on the left contains 976 points after the steering phase, while the same region in the plot on the right contains only 595 points. The line chart in the foreground shows how the Precision metric increased during the steering phase from about 0.1 on average to about 0.45.

dom sampling shows that, according to the actual sample size (about 400,000 random items), the user-selected area has a very low density, making more evident the advantage of the steering phase.

8.6.2 *Steering-by-Example on Dimensionality Reductions*

We further assessed the generality of steering-by-example regarding the visual encoding. While the progressive visualization used for benchmarks placed the data along two computed axes, in dimensionality reduction the 2D position of a data item is computed from multiple dimensions, to make high-dimensional features of the data interpretable to the user [35]. In other words, the mapping function f is implemented as a complex algorithm. In particular, we used the RadViz dimensionality reduction method [4], which uses a spring-based model to compute the position of items along n dimension in a radial layout. Using the NYC taxis dataset we introduced above in Section 8.6.1 on a RadViz-viewer for ProSteer, we let the progression run under the same conditions as in the benchmarks and selected a region of interest. We found that in comparison with our benchmarks on the scatter plot, steering-by-example performed worse in terms of the average Precision, yet still produced a noticeable increase in data in the selected region from a (subjective) user perspective when compared to the baseline scenario. For instance, Figure 45 shows screenshots from an exemplar run after the steering phase ended, indicating how the progression using steering-by-example produces almost double the number of items in the selected regions, with an average Precision of around 0.45 during the steering phase. Nevertheless, these results are only a preliminary indication of the utility of steering-by-example when combined with dimensionality reduction techniques, and further evaluations are necessary to measure (and improve) its performance in that regard.

8.7 USER STUDY

To assess the usability of steering-by-example, we conducted a user study with VA experts, to evaluate how users would apply the approach to address an analysis goal, and what their experience is in doing so. Note, that the purpose of this study was to collect expert feedback on steering-by-example as a conceptual approach, rather than to evaluate ProSteer as a practical visual analytics tool. We first report on the setup and procedure used, followed by an evaluation of collected results.

8.7.1 *Setup*

We recruited five participants (3m and 2f, aged between 24 and 33) from our departments, who actively work in the research field of visual analytics as Ph.D. students, PostDocs, and research assistants. On a 7-point Likert scale, participants reported strong expertise with PVA (median: 6), a strong familiarity with data science (median: 6), and with visualization (median: 6). Most were familiar with computational steering (median: 4). User studies lasted between 40 and 80 minutes and were conducted in a quiet environment, using either the participants' or experimenters' laptop computers.

8.7.2 *Procedure*

The experiment was split into four distinct phases:

POSITIONING QUESTIONNAIRE: In this phase, participants were tasked to read and sign a consent form, followed by a questionnaire concerning their expertise on the subjects of steering-by-example. The questionnaire consisted of three positioning questions (Age, Gender, and Role) and of five questions for self-assessing their expertise, based on a 7-point Likert scale (Q4: How familiar are you with data analysis?, Q5: How familiar are you with visual analytics?, Q6: How familiar are you with visualization?, Q7: How familiar are you with what is called "progressive visual analytics"?, and Q8: How familiar are you with what is called "computational steering"?). Participants had up to 15 minutes to complete this phase.

INTRODUCTION TO PROSTEER: In this phase, participants followed a live demonstration of ProSteer. We illustrated its main functionalities and the available user interactions. Participants were able to interject at any moment and ask questions about any part of the approach and the software prototype that remained unclear. The demonstration lasted from a minimum of 20 minutes up to the time to which participants explicitly confirmed to have understood everything and did not have additional questions.

interactive usage of prosteer: In this phase, we tasked participants to load ProSteer in their environment (participants were provided materials and assistance in setting up the ProSteer prior to the user study), load the NYC taxi and AirBnB datasets, and solve the following task: "Find all apartments that are close by, which in this neighborhood allow you to save as much money as possible." This task and its intentionally vague formulation was designed to maximize exploration of alternatives in order to find a potential solution, implicitly asking for steering support to explore these alternatives faster. There was no fixed time span allotted to this phase, but a suggested time of 20 minutes was communicated to the participants. Participants could end this phase prior to this limit, if they found their analysis results to be sufficient.

evaluation questionnaire: Finally, in this phase the participants were asked to fill out an evaluation questionnaire on their usage experience of ProSteer. The questionnaire was composed of eight open questions (Q9: Do you understand what is going on in the interface? What questions arise?, Q10: Change blindness of new points arriving in the interface during progression. Is that confusing to you?, Q11: Does focusing the progression on a certain region help?, Q12: Do you find it challenging that you do not fully understand the steering mechanism (i.e., that it is a black box)?, Q13: What are your considerations with respect to the shape of the selection? Is it good? Is it enough?, Q14: What are your considerations with respect to making multiple selections?, Q15: Steering means biasing the progression towards a certain part of the data. What are your thoughts on that, having used ProSteer for a bit?, and Q16: Can you see where this would fit into your data analysis workflow? Is this helpful to you?. A final question (Q19) was asking for any additional notes or comments from the participants. The time dedicated to this phase was 20 minutes.

8.7.3 *Results*

user behavior: In terms of the usability of steering-by-example, all participants in our user study successfully used our approach to steer the progression towards interesting data and were able to complete the task. We generally observed an *iterative* interaction procedure: First, participants monitored data arriving in the progressive visualization, then they identified a region of interest, and afterward steered the progression towards that region by making a selection, returning to the observation step. This procedure matches well with the states of our steering-by-example approach, as discussed in Section 8.3. We did, however, also observe deviations from this general procedure. For instance, sometimes participants (E1, E2) began at the second stage, selecting a region of interest before any data had arrived and thereby skipping the initial non-steering phase in our approach. Essentially, this behavior likens the steering based on a 1-to-1 mapping as used by the state-of-the-art approaches we reported on in Section 8.2. This similarity suggests the need for future experiments, to determine the impact of different steering methods on the user experience, e.g., whether participants would actually notice that the steering mechanism differs between analysis scenarios. Other deviations we observed were participants (E1, E2) trying to make multiple selections corresponding to multiple regions of interest in the data. This behavior is characteristic for PVA, as analysts at any point in time can choose to adjust the ongoing computation to new insights gained from the latest data [78]. Thus, to further improve the usability of steering-by-example, future iterations on the approach should consider this flexibility as a design goal. A challenge that needs to be considered is whether to treat items from different selections together or separately during the training phase, i.e., whether to train one or multiple models. We also observed participants being unsure about the current phase of the steering. For instance, E3 at one point did not consider the steering phase indicator when looking at the interface, suggesting to make the current phase of the approach more prominent in the user interface. One adjustment to this end could be to adjust the visual style of the selection box, such that it changes color based on the current phase and its border thickness based on the current Precision.

USER EXPERIENCE: All participants reported that they found steering helpful in general, and could see the potential for long-running analyses that PVA is used for, in particular if users have a clear goal in mind. Expert E2 for example stated that "it is intuitive to support steering if the user has identified a region they are interested in" and E1 found it "helpful when you want to test a hypothesis on a very large dataset", while E4 noted that it might not be helpful for pure exploration of the data without a clear goal. When prompted, participants stated that highlighting the most recent points is useful for overcoming change blindness, and that the matrix in the background was useful in indicating that steering affected the sampling of the data in that region. All participants generally found ProSteer's interface helpful and appropriate for understanding the impact that steering has on the analysis. Some participants (E3, E4, E5) however noted that the encoding of recent points on the Taxi dataset could be improved, as points inside the selection became difficult to identify at later stages, and they reported on the issue of overplotting. They also suggested extending ProSteer to allow for adjusting the selected region during or after the steering phase to refine the selected region (E1, E2, E5).

Overall, participants agreed that the steering-by-example approach could potentially be of use for their own data analyses. These preliminary interviews provide a first impression of the utility and accessibility of steering-by-example. While the results are generally positive, our interviews can only serve as initial impressions and further evaluations are necessary to this end

8.8 DISCUSSION

Steering-by-example allows prioritizing data subspaces for progressive computations for a variety of use cases. In this section, we want to take a step back and discuss the generality of the approach, looking at implicit assumptions and limitations.

8.8.1 *Implicit Assumptions of Steering-by-Example*

SELECTIONS ARE SENSIBLE: Steering-by-example requires a set of data items as input, which are assumed to be of interest to the user. The underlying assumption here is that these items share some characteristic in data space that makes them interesting compared to the rest of the data, and that steering-by-example can identify these characteristics and translate them into an SQL query. Only if the selection of items is thus sensible can the query produced by steering-by-example retrieve other items sharing these interesting characteristics. A challenge is that if the selection is arbitrary, the decision tree would learn a similarly arbitrary set of decision rules that yields items that are potentially not interesting to the user. That assumption is based on the idea that an overarching goal of analyzing data through visualization is to use sensible mappings to encode that data, such that similar values in data space are represented close to each other in view space. Therefore, visualizations inherently support users when interactively creating sensible selections. An exception to this rule is bubble charts, in which the position is decided purely based on a circle packing algorithm, rather than the data. Beyond the visual encoding, we can also support the user through statistical approaches like active learning [104], automatically suggesting rendered items that may be of interest to the user based on data characteristics that distinguish them in the dataset.

SELECTIONS ARE PERMANENT: In addition to selections being sensible, another implicit assumption to steering-by-example is that selections are permanent, i.e., an item of the data that lies inside the selection will remain inside the selection until the steering phase concludes. This assumption is based on the idea that users are expected to wait until the progressive visualization has stabilized (i.e., the "quality" of the visualization is high enough [5]), before they form an interest in the data. In our implementation and evaluation, this assumption manifests in that items do not change their position in the scatter plot, since axis extents were known upfront, and thus items are static, while the selection itself cannot be moved around by the user during steering. In some progressive scenarios, item locations are however not stable and thus items move into and out of the selection. For example, in progressive dimensionality reduction methods like incremental PCA discussed by Ross et al. [101], the location of rendered items needs to be updated as the model is trained on increments of the data, thus the computed axes change. Therefore, items that were previously inside the selection are potentially located outside that selection after updating positions. To address this, we in essence need to monitor the selection's "fluctuation" and retrain the decision tree, once the number of items in the selection that changed reaches a threshold.

8.8.2 *Limitations of Steering-by-Example in PVA*

STEERING MAY CAUSE CONTROL BIAS: A general challenge in PVA is appropriately representing the uncertainty of the visualization data caused by the use of partial results. When also allowing users to steer the computation – which causes some regions of the data to be more "certain" than others – this representation becomes more complicated. When steering a progressive computation towards a region of data the user selected as interesting using any steering mechanism, the visualization is more "refined" in those regions, which can mislead users in their interpretation of the overall data distribution. The visualization is no longer "evenly incomplete", and so the visualized data needs to be interpreted under consideration of that unevenness, which users need to be (made) aware of. Micallef et al. call this phenomenon "control bias" [78]. Even though recent work by Procopio et al. has shown that the effect of control bias in PVA through steering usually does not affect users' performance, but in fact increases their certainty in the conclusions they draw [94], implementing steering mechanisms like steering-by-example into a visual analytics system nevertheless requires careful consideration of how completeness is encoded. In ProSteer, we use the heatmap in the background of the data to indicate the difference in sampling caused by steering, to immediately inform users about these effects. Furthermore, by highlighting the location of the latest data, changes in the data spread per chunk during steering become clearly noticeable.

STEERING REQUIRES ADEQUATE DATA DENSITY: In order for steering-by-example to generate a query that yields appropriate results, our approach needs a certain number of interesting representatives as input. This number has both an upper and a lower bound. If the number of items in the selection is too low and will never reach the threshold of items needed for training even if the progression completes, the decision tree cannot adequately capture the characteristics that make data interesting, as evident in some test cases of our benchmarks with sparse selections (see Section 8.4). Consequently, if the number of items is too high, training the decision tree classifier can take too long. A potential solution to the former challenge that we discuss in Section 8.3.3, is to artificially increase the scope of the selection to also include neighboring regions. While this reduces the accuracy of the steering query, it can nevertheless increase the overall relevance of the retrieved items. For very sparse regions however, even widening the scope of the selection in this way may not be sufficient, meaning that steering-by-example simply cannot support the user. For example, when users are interested in finding outliers and thus select a single item, extending that selection may select enough other data to trigger the training, yet it is exactly those data that the user was *not* interested in when selecting the outlier. Vice versa, a solution addressing selections with too many items in them is to train the decision tree on a smaller sample. This can potentially reduce the accuracy of the steering query, yet we can avoid training times becoming too long. Supporting this approach, prior work [85] as well as our own tests (see Figure 38) suggest that decision trees do not necessarily benefit from having more data in the training sets, and so sampling may be sufficient.

8.9 CONCLUSION AND FUTURE WORK

In this paper, we presented steering-by-example, a novel approach to prioritizing data subspaces during progressive computations that are of interest to the user, based on a user selection in view space. We evaluated our approach with a series of benchmarks, which show steering-by-example significantly outperforming random uniform sampling in terms of Precision and Recall for relevant items in the selected view region. To demonstrate steering-by-example, we provide the open-source visual benchmarking interface ProSteer.

Beyond extending the benchmarks towards more generalizable results (see Section 8.8), further questions arise for further developing the steering-by-example approach conceptually.

One field of future work is to use a continuous degree-of-interest function defined over the items in a selection rather than a strictly binary distinction between inside/relevant and outside/irrelevant. Steering-by-example currently expects that all data items inside the view selection are of equal interest to the user. However, view space selections are usually made in a fuzzy, approximate manner, rather than with surgical Precision. That means that the training data for the decision tree contains both items that are of high interest to the user as well as items that are of lesser interest. Yet, both contribute equally to the model. We want to address this in future work, by defining a degree-of-interest function over the data inside a selection, which assigns a continuous value to each data item instead of making a binary distinction. This function could for example consider how close to the center and how close to the boundary of the selection a data item lies. Then, we want to use regression trees [8] trained on these continuous values for building the model of the approximate inverse mapping function from view to data space. By increasing the expressivity of the model, the idea is that the performance of steering-by-example can be further improved.

We see a second potential field for future work in adapting steering-by-example for *iterative* progressions. In the introduction section, we have discussed PVA as a way for bringing the user into the loop of long-running computations through iterative or incremental approaches. We presented steering-by-example for *incremental* computations, in which the result of the computation progressively includes a larger subspace of a dataset, eventually ending up at the same result as were the computation run on the entire dataset. This naturally begs the question, how to adapt the approach for *iterative* computations such as node-link layouts or iterative clustering, where a computation instead progressively refines the result computed over the entire dataset, eventually converging towards a stable output. One potential steering-by-example approach could be described as focused refinement, i.e., instead of iterating over the entire dataset, the computation could prioritize subspaces similar to those selected by the user, producing a stable result first for these spaces before refining the rest of the data. Another way to involve the user in iterative computation is to let them provide examples of what they expect the final computation to look like. For instance, in the case of clustering, such an approach could be utilized as initialization of the computation: The user could manually define the clusters for a subset of the data based on their domain knowledge, and the system could then transfer this to the entire dataset, assigning similar data to similar clusters as in the user's selection. For both adaptions of steering-by-example, the research challenge lies in solving the inherent algorithmic and computational challenges.

Finally, while for practical reasons our benchmark used only a single box as user selection shape, it would be interesting to explore both multiple selections and other selection shapes like circles and lassos. Indeed, while our approach is generally independent of the way in which data items for the training phase are gathered, investigating not rectangular shapes or multiple selections in view space could both confirm the result presented in this paper for the box-based selector and push for exploring different steering methods. That can lead to tailoring decision trees for specific selection shapes or using multiple decision tree models, each trained on one selection of the view space. Then, the steering query could be constructed by disjuncting the individual predicates extracted from each tree. Yet, the implications and side-effects of these solutions need to be further investigated.

ACKNOWLEDGEMENTS

We are indebted to Heidrun Schumann, Georg Fuchs, Chris Weaver, and Davide Mottin with whom we discussed the steering-by-example problem in the inception phase of this work.

Part III

APPENDIX

BIBLIOGRAPHY

[1] James Abello, Steffen Hadlak, Heidrun Schumann, and Hans-Jörg Schulz. "A Modular Degree-of-Interest Specification for the Visual Analysis of Large Dynamic Networks." In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2014), pp. 337–350. ISSN: 1077-2626. DOI: 10.1109/TVCG.2013.109.

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. "BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data." In: *Proc. of EuroSys*. ACM, 2013, pp. 29–42. DOI: 10.1145/2465351.2465355.

[3] S. Albers. "Online algorithms: A Survey." In: *Mathematical Programming* 97 (2003), pp. 3–26. DOI: 10.1007/s10107-003-0436-0.

[4] Marco Angelini, Graziano Blasilli, Simone Lenti, Alessia Palleschi, and Giuseppe Santucci. "Effectiveness Error: Measuring and Improving RadViz Visual Effectiveness." In: *IEEE Transactions on Visualization and Computer Graphics* (2021), pp. 1–1. DOI: 10.1109/TVCG.2021.3104879.

[5] Marco Angelini, Thorsten May, Giuseppe Santucci, and Hans-Jörg Schulz. "On Quality Indicators for Progressive Visual Analytics." In: *Proc. of EuroVA*. Eurographics Association, 2019, pp. 025–029. DOI: 10.2312/eurova.20191120.

[6] Marco Angelini, Giuseppe Santucci, Heidrun Schumann, and Hans-Jörg Schulz. "A Review and Characterization of Progressive Visual Analytics." In: *Informatics* 5.3 (2018), 31:1–31:27. ISSN: 2227-9709. DOI: 10.3390/informatics5030031.

[7] F. J. Anscombe. "Graphs in Statistical Analysis." In: *The American Statistician* 27.1 (1973), pp. 17–21. DOI: 10.1080/00031305.1973.10478966.

[8] Chidanand Apté and Sholom Weiss. "Data mining with decision trees and decision rules." In: *Future Generation Computer Systems* 13.2 (1997), pp. 197–210. DOI: 10.1016/S0167-739X(97)00021-6.

[9] B. P. Welford. "Note on a Method for Calculating Corrected Sums of Squares and Products." In: *Technometrics* 4.3 (1962), pp. 419–420. DOI: 10.1080/00401706.1962.10490022.

[10] Sriram Karthik Badam, Niklas Elmqvist, and Jean-Daniel Fekete. "Steering the Craft: UI Elements and Visualizations for Supporting Progressive Visual Analytics." In: *Computer Graphics Forum* 36.3 (2017), pp. 491–502. DOI: 10.1111/cgf.13205.

[11] Wesam Barbakh and Colin Fyfe. "Online Clustering Algorithms." In: *International Journal of Neural Systems* 18.03 (2008). PMID: 18595148, pp. 185–194. DOI: 10.1142/S0129065708001518.

[12] Leilani Battle, Remco Chang, and Michael Stonebraker. "Dynamic Prefetching of Data Tiles for Interactive Visualization." In: *Proc. of SIGMOD*. New York, NY, USA: ACM, 2016, pp. 1363–1375. DOI: 10.1145/2882903.2882919.

[13] Lukas Berg, Tobias Ziegler, Carsten Binnig, and Uwe Röhm. "ProgressiveDB: Progressive Data Analytics as a Middleware." In: *Proc. VLDB Endow.* 12.12 (2019), pp. 1814–1817. DOI: 10.14778/3352063.3352073.

[14] M. Bostock, V. Ogievetsky, and J. Heer. "D^3 Data-Driven Documents." In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2301–2309. DOI: 10.1109/TVCG.2011.185.

[15] Leo Breiman, Jerome Friedmann, and Charles J. Stone. *Classification and Regression Trees*. Monterey: Chapman and Hall/CRC, 1984.

[16] Frederick P. Brooks. "The Computer Scientist as Toolsmith II." In: *Communications of the ACM* 39.3 (1996), pp. 61–68. DOI: 10.1145/227234.227243.

[17] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision To Think*. Academic Press, 1999.

[18] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. "The Information Visualizer, an Information Workspace." In: *Proc. of CHI*. ACM, 1991, pp. 181–186. DOI: 10.1145/108844.108874.

[19] Stuart Card. "The Human-Computer Interaction Handbook." In: ed. by Julie E. Jacko and Andrew Sears. Lawrence Erlbaum Associates, 2003. Chap. Information Visualization, pp. 544–583.

[20] Davide Ceneda, Theresia Gschwandtner, Thorsten May, Silvia Miksch, Hans-Jörg Schulz, Marc Streit, and Christian Tominski. "Characterizing Guidance in Visual Analytics." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 111–120. DOI: 10.1109/tvcg.2016.2598468.

[21] Badrish Chandramouli, Jonathan Goldstein, and Abdul Quamar. "Scalable Progressive Analytics on Big Data in the Cloud." In: *Proc. of VLDB Endow.* 6.14 (2013), pp. 1726–1737. DOI: 10.14778/2556549.2556557.

[22] Min Chen, Luciano Floridi, and Rita Borgo. "What Is Visualization Really For?" In: *The Philosophy of Information Quality*. February. 2014, pp. 75–93. DOI: 10.1007/978-3-319-07121-3_5.

[23] Xin Chen, Jian Zhang, Chi-Wing Fu, Jean-Daniel Fekete, and Yunhai Wang. "Pyramid-based Scatterplots Sampling for Progressive and Streaming Data Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* (2021), pp. 1–11. DOI: 10.1109/TVCG.2021.3114880.

[24] William S. Cleveland and Robert McGill. "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods." In: *Journal of the American Statistical Association* 79.387 (1984), pp. 531–554. DOI: 10.1080/01621459.1984.10478080.

[25] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches." In: *Foundations and Trends in Databases* 4.1–3 (2011), pp. 1–294. ISSN: 1931-7883. DOI: 10.1561/1900000004.

[26] Mihaly Csikszentmihalyi. *Flow: The psychology of optimal experience*. Harper & Row, 1990.

[27] Z. Cui, J. Kancherla, H. C. Bravo, and N. Elmqvist. "Sherpa: Leveraging User Attention for Computational Steering in Visual Analytics." In: *Proc. of VDS*. IEEE, 2019, pp. 48–57. DOI: 10.1109/VDS48975.2019.8973384.

[28] Ismail Demir, Christian Dick, and Rüdiger Westermann. "Multi-Charts for Comparative 3D Ensemble Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2694–2703. DOI: 10.1109/TVCG.2014.2346448.

[29] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. "Explore-by-Example: An Automatic Query Steering Framework for Interactive Data Exploration." In: *Proc. of SIGMOD*. New York, NY, USA: ACM, 2014, pp. 517–528. DOI: 10.1145/2588555.2610523.

[30] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. "Sample + Seek: Approximating Aggregates with Distribution Precision Guarantee." In: *Proc. of SIGMOD*. ACM, 2016, pp. 679–694. DOI: 10.1145/2882903.2915249.

[31] Geoffrey Ellis, Enrico Bertini, and Alan Dix. "The Sampling Lens: Making Sense of Saturated Visualisations." In: *Extended Abstract Proc. of CHI*. Portland, OR, USA: ACM, 2005, pp. 1351–1354. ISBN: 1595930027. DOI: 10.1145/1056808.1056914.

[32] Niklas Elmqvist and Jean-Daniel Fekete. "Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines." In: *IEEE Transactions on Visualization and Computer Graphics* 16.3 (2010), pp. 439–454. DOI: 10.1109/TVCG.2009.84.

[33] Niklas Elmqvist, Andrew Vande Moere, Hans-Christian Jetter, Daniel Cernea, Harald Reiterer, and T. J. Jankun-Kelly. "Fluid interaction for information visualization." In: *Information Visualization* 10.4 (2011), pp. 327–340. DOI: 10.1177/1473871611413180.

[34] Alex Endert, M. Shahriar Hossain, Naren Ramakrishnan, Chris North, Patrick Fiaux, and Christopher Andrews. "The human is the loop: new directions for visual analytics." In: *Journal of Intelligent Information Systems* 43 (2014), pp. 411–435. DOI: 10.1007/s10844-014-0304-9.

[35] Mateus Espadoto, Gabriel Appleby, Ashley Suh, Dylan Cashman, Mingwei Li, Carlos E. Scheidegger, Erik Wesley Anderson, Remco Chang, and Alexandru Cristian Telea. "UnProjection: Leveraging Inverse-Projections for Visual Analytics of High-Dimensional Data." In: *IEEE Transactions on Visualization and Computer Graphics* (2021), pp. 1–15. DOI: 10.1109/TVCG.2021.3125576.

[36] S. I. Fabrikant, D. R. Monteilo, and D. M. Mark. "The distance-similarity metaphor in region-display spatializations." In: *IEEE Computer Graphics and Applications* 26.4 (2006), pp. 34–44. DOI: 10.1109/MCG.2006.90.

[37] Jean-Daniel Fekete. "ProgressiVis: a Toolkit for Steerable Progressive Analytics and Visualization." In: *Proc. of the Workshop on Data Systems for Interactive Analysis*. Chicago, United States, 2015, pp. 1–5. URL: https://hal.inria.fr/hal-01202901.

[38] Jean-Daniel Fekete, Qing Chen, Yuheng Feng, and Jonas Renault. "Practical Use Cases for Progressive Visual Analytics." In: *DSIA 2019 - 4th Workshop on Data Systems for Interactive Analysis*. Vancouver, Canada, Oct. 2019, pp. 1–5. URL: https://hal.inria.fr/hal-02342944.

[39] Jean-Daniel Fekete, Danyel Fisher, Arnab Nandi, and Michael Sedlmair (Eds.) "Progressive Data Analysis and Visualization." In: *Dagstuhl Reports* 8.10 (2018), pp. 1–40. DOI: 10.4230/DagRep.8.10.1.

[40] Jean-Daniel Fekete, Danyel Fisher, and Michael Sedlmaier, eds. *Progressive Data Analysis – Roadmap and Research Agenda*. (in preparation). Springer, 2022.

[41] Jean-Daniel Fekete and Romain Primet. "Progressive Analytics: A Computation Paradigm for Exploratory Data Analysis." In: *CoRR* (2016). eprint: 1607.05162. URL: http://arxiv.org/abs/1607.05162.

[42] Nivan Ferreira, Jorge Poco, Huy T. Vo, Juliana Freire, and Cláudio T. Silva. "Visual Exploration of Big Spatio-Temporal Urban Data: A Study of New York City Taxi Trips." In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2149–2158. DOI: 10.1109/TVCG.2013.226.

[43] Danyel Fisher, Igor Popov, Steven Drucker, and m.c. schraefel. "Trust Me, i'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster." In: *Proc. of CHI*. ACM, 2012, pp. 1673–1682. DOI: 10.1145/2207676.2208294.

[44] Yaniv Frishman and Ayellet Tal. "Online Dynamic Graph Drawing." In: *IEEE Transactions on Visualization and Computer Graphics* 14.4 (2008), pp. 727–740. DOI: 10.1109/TVCG.2008.11.

[45] G. W. Furnas. "The FISHEYE view: a new look at structured files." In: *Readings in Information Visualization: Using Vision to Think*. Ed. by S. K. Card, J. D. Mackinlay, and B. Shneiderman. Morgan Kaufmann Publishers, 1981, pp. 312–330.

[46] G. W. Furnas. "Generalized Fisheye Views." In: *Proc. of CHI*. ACM, 1986, pp. 16–23. DOI: 10.1145/22627.22342.

[47] Kiran Gadhave, Jochen Görtler, Zach Cutler, Carolina Nobre, Oliver Deussen, Miriah Meyer, Jeff M. Phillips, and Alexander Lex. "Predicting intent behind selections in scatterplot visualizations." In: *Information Visualization* 20.4 (2021), pp. 207–228. DOI: 10.1177/14738716211038604.

[48] Baohua Gu, Bing Liu, Feifang Hu, and Huan Liu. "Efficiently Determining the Starting Sample Size for Progressive Sampling." In: *Proc. of ECML*. Springer, 2001, pp. 192–202. DOI: 10.1007/3-540-44795-4_17.

[49] Jeffrey Heer and Maneesh Agrawala. "Software Design Patterns for Information Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 853–860. DOI: 10.1109/TVCG.2006.178.

[50] Jeffrey Heer and Stuart K. Card. "DOITrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data." In: *Proc. of AVI*. Gallipoli, Italy: ACM, 2004, pp. 421–424. ISBN: 1581138679. DOI: 10.1145/989863.989941.

[51] 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." In: *Computer* 32.8 (1999), pp. 51–59. DOI: 10.1109/2.781635.

[52] M. Hogräfer, J. Burkhardt, and H.-J. Schulz. "A Pipeline for Tailored Sampling for Progressive Visual Analytics." In: *Proc. of EuroVA*. Eurographics Association, 2022.

[53] Marius Hogräfer, Marco Angelini, Giuseppe Santucci, and Hans-Jörg Schulz. "Steering-by-Example for Progressive Visual Analytics." In: *ACM Transactions on Intelligent Systems and Technology* 13.6 (2022). To appear, 96:1–96:26. DOI: 10.1145/3531229.

[54] Michael Hohenstein. "Progressive Indexing for Interactive Analytics." In: *Proc. of GI-Workshop on Foundations of Databases*. 2021, pp. 1–7. URL: http://ceur-ws.org/Vol-3075/paper5.pdf.

[55] Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. "Online Learning: A Comprehensive Survey." In: *CoRR* abs/1802.02871 (2018). URL: http://arxiv.org/abs/1802.02871.

[56] Pedro Holanda, Mark Raasveldt, Stefan Manegold, and Hannes Mühleisen. "Progressive Indexes: Indexing for Interactive Data Analysis." In: *Proc. VLDB Endowments* 12.13 (2019), pp. 2366–2378. DOI: 10.14778/3358701.3358705.

[57] Meishan Hu, Aixin Sun, and Ee-Peng Lim. "Event Detection with Common User Interests." In: *Proc. of WIDM*. ACM, 2008, pp. 1–8. DOI: 10.1145/1458502.1458504.

[58] Peter J. Huber. "Robust Statistics." In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Springer, 2011, pp. 1248–1251. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_594. URL: https://doi.org/10.1007/978-3-642-04898-2_594.

[59] J. Jo, J. Seo, and J. Fekete. "PANENE: A Progressive Algorithm for Indexing and Querying Approximate k-Nearest Neighbors." In: *IEEE Transactions on Visualization and Computer Graphics* 26.2 (2020), pp. 1347–1360. DOI: 10.1109/TVCG.2018.2869149.

[60] Jaemin Jo. "Designing Progressive Visualization Systems for Exploring Large-scale Data." PhD thesis. Seoul National University, Feb. 2020.

[61] Jaemin Jo, Sehi L'Yi, Bongshin Lee, and Jinwook Seo. "ProReveal: Progressive Visual Analytics With Safeguards." In: *IEEE Transactions on Visualization and Computer Graphics* 27.7 (2021), pp. 3109–3122. DOI: 10.1109/TVCG.2019.2962404.

[62] Jaemin Jo, Jinwook Seo, and Jean-Daniel Fekete. "A progressive k-d tree for approximate k-nearest neighbors." In: *Proc. of DSIA*. 2017, pp. 1–5. DOI: 10.1109/DSIA.2017.8339084.

[63] D. A. Keim, C. Panse, M. Sips, and S. C. North. "PixelMaps: a new visual data mining approach for analyzing large spatial data sets." In: *Proc. of Data Mining*. IEEE, 2003, pp. 565–568. DOI: 10.1109/ICDM.2003.1250978.

[64] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. "Visual Analytics: Definition, Process, and Challenges." In: *Information Visualization: Human-Centered Issues and Perspectives*. Ed. by Andreas Kerren, John T. Stasko, Jean-Daniel Fekete, and Chris North. Springer, 2008, pp. 154–175. DOI: 10.1007/978-3-540-70956-5_7. URL: https://doi.org/10.1007/978-3-540-70956-5_7.

[65] Hyung-Kwon Ko, Jaemin Jo, and Jinwook Seo. "Progressive Uniform Manifold Approximation and Projection." In: *Proc. of EuroVis Short Papers*. Eurographics Association, 2020, pp. 133–137. DOI: 10.2312/evs.20201061.

[66] Marcel Köster and Antonio Krüger. "Screen Space Particle Selection." In: *Proc. of CGVC*. Manchester, United Kingdom: Eurographics Association, 2018, pp. 61–69. DOI: 10.2312/cgvc.20181208.

[67] Bum Chul Kwon, Janu Verma, Peter J. Haas, and Çagatay Demiralp. "Sampling for Scalable Visual Analytics." In: *IEEE Computer Graphics and Applications* 37.1 (2017), pp. 100–108. DOI: 10.1109/MCG.2017.6.

[68] J. K. Li and K. Ma. "P5: Portable Progressive Parallel Processing Pipelines for Interactive Data Analysis and Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), pp. 1151–1160. DOI: 10.1109/TVCG.2019.2934537.

[69] Xiaolei Li, Jiawei Han, Zhijun Yin, Jae-Gil Lee, and Yizhou Sun. "Sampling Cube: A Framework for Statistical Olap over Sampling Data." In: *Proc. of SIGMOD*. ACM, 2008, pp. 779–790. DOI: 10.1145/1376616.1376695.

[70] Robert van Liere, Jurriaan D. Mulder, and Jarke J. van Wijk. "Computational steering." In: *Future Generation Computer Systems* 12.5 (1997), pp. 441–450. DOI: 10.1016/S0167-739X(96)00029-5.

[71] Yang Liu, Eunice Jun, Qisheng Li, and Jeffrey Heer. "Latent Space Cartography: Visual Analysis of Vector Space Embeddings." In: *Computer Graphics Forum* 38.3 (2019), pp. 67–78. DOI: 10.1111/cgf.13672.

[72] Zhicheng Liu and Jeffrey Heer. "The Effects of Interactive Latency on Exploratory Visual Analysis." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2122–2131. DOI: 10.1109/TVCG.2014.2346452.

[73] S. Lloyd. "Least squares quantization in PCM." In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[74] Sebastian Loeschcke, Marius Hogräfer, and Hans-Jörg Schulz. "Progressive Parameter Space Visualization for Task-Driven SAX Configuration." In: *Proc. of EuroVA*. Ed. by Cagatay Turkay and Katerina Vrotsou. Eurographics Association, 2020, pp. 43–47. ISBN: 978-3-03868-116-8. DOI: 10.2312/eurova.20201085.

[75] Viktor Losing, Barbara Hammer, and Heiko Wersing. "KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift." In: *Proc. of ICDM*. 2016, pp. 291–300. DOI: 10.1109/ICDM.2016.0040.

[76] Viktor Losing, Barbara Hammer, and Heiko Wersing. "Incremental on-line learning: A review and comparison of state of the art algorithms." In: *Neurocomputing* 275 (2018), pp. 1261–1274. DOI: 10.1016/j.neucom.2017.06.084.

[77] Jock Mackinlay. "Automating the Design of Graphical Presentations of Relational Information." In: *ACM Transactions on Graphics* 5.2 (1986), pp. 110–141. DOI: 10.1145/22949.22950.

[78] Luana Micallef, Hans-Jörg Schulz, Marco Angelini, Michaël Aupetit, Remco Chang, Jörn Kohlhammer, Adam Perer, and Giuseppe Santucci. "The Human User in Progressive Visual Analytics." In: *Proc. of EuroVis Short Papers*. Eurographics Association, 2019, pp. 19–23. DOI: 10.2312/evs.20191164.

[79] Robert B. Miller. "Response Time in Man-Computer Conversational Transactions." In: *Proc. of AFIPS*. ACM, 1968, pp. 267–277. DOI: 10.1145/1476589.1476628.

[80] Dominik Moritz, Danyel Fisher, Bolin Ding, and Chi Wang. "Trust, but Verify: Optimistic Visualizations of Approximate Queries for Exploring Big Data." In: *Proc of CHI*. Denver, Colorado, USA: ACM, 2017, pp. 2904–2915. ISBN: 9781450346559. DOI: 10.1145/3025453.3025456.

[81] Davide Mottin, Alice Marascu, Senjuti Basu Roy, Gautam Das, Themis Palpanas, and Yannis Velegrakis. "IQR: An Interactive Query Relaxation System for the Empty-Answer Problem." In: *Proc. of SIGMOD*. New York, NY, USA: ACM, 2014, pp. 1095–1098. DOI: 10.1145/2588555.2594512.

[82] Thomas Mühlbacher, Harald Piringer, Samuel Gratzl, Michael Sedlmair, and Marc Streit. "Opening the black box: Strategies for increased user involvement in existing algorithm implementations." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 1643–1652. DOI: 10.1109/TVCG.2014.2346578.

[83] T. Munzner. "A Nested Model for Visualization Design and Validation." In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 921–928. DOI: 10.1109/TVCG.2009.111.

[84] Donald Norman. *Things that make us smart: defending human attributes in the age of the machine*. Addison-Wesley, 1993.

[85] Tim Oates and David Jensen. “The Effects of Training Set Size on Decision Tree Complexity.” In: *Proc. of ICML*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 254–262. DOI: 10.5555/645526.657136.

[86] Frank Olken and Doron Rotem. “Random sampling from database files: A survey.” In: *Statistical and Scientific Database Management*. Springer, 1990, pp. 92–111. DOI: 10.1007/3-540-52342-1_23.

[87] Melih Onus, Andrea Richa, and Christian Scheideler. “Linearization: Locally Self-Stabilizing Sorting in Graphs.” In: *Proc. of the ALENEX Workshop*. SIAM, 2007, pp. 99–108. DOI: 10.1137/1.9781611972870.10.

[88] OpenStreetMap. *Mountain Peaks Data*. downloaded 01-May-2021. URL: https://wiki.openstreetmap.org/wiki/Tag:natural=peak.

[89] Y. Park, M. Cafarella, and B. Mozafari. “Visualization-aware sampling for very large databases.” In: *Proc. of ICDE*. IEEE, 2016, pp. 755–766. DOI: 10.1109/ICDE.2016.7498287.

[90] Ameya Patil, Gaëlle Richer, Christopher Jermaine, Dominik Moritz, and Jean-Daniel Fekete. “Studying Early Decision Making with Progressive Bar Charts.” In: *IEEE Transactions on Visualization and Computer Graphics* (2022). to be published, pp. 1–11. URL: https://hal.inria.fr/hal-03738461.

[91] Philippe Pierre Pébay. *Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments.* Tech. rep. U.S. Department of Energy – Office of Scientific and Technical Information, Sept. 2008. DOI: 10.2172/1028931.

[92] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python.” In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[93] N. Pezzotti, B. P. F. Lelieveldt, L. v. d. Maaten, T. Höllt, E. Eisemann, and A. Vilanova. “Approximated and User Steerable tSNE for Progressive Visual Analytics.” In: *IEEE Transactions on Visualization and Computer Graphics* 23.7 (2017), pp. 1739–1752. DOI: 10.1109/TVCG.2016.2570755.

[94] M. Procopio, A. Mosca, C. Scheidegger, E. Wu, and R. Chang. “Impact of Cognitive Biases on Progressive Visualization.” In: *IEEE Transactions on Visualization and Computer Graphics* (2021), pp. 1–13. DOI: 10.1109/TVCG.2021.3051013.

[95] Marianne Procopio, Carlos Scheidegger, Eugene Wu, and Remco Chang. "Selective Wander Join: Fast Progressive Visualizations for Data Joins." In: *Informatics* 6.1 (2019), pp. 1–21. ISSN: 2227-9709. DOI: 10.3390/informatics6010014.

[96] Foster Provost, David Jensen, and Tim Oates. "Efficient Progressive Sampling." In: *Proc. of SIGKDD*. San Diego, California, USA: ACM, 1999, pp. 23–32. ISBN: 1581131437. DOI: 10.1145/312129.312188.

[97] Sajjadur Rahman, Maryam Aliakbarpour, Ha Kyung Kong, Eric Blais, Karrie Karahalios, Aditya Parameswaran, and Ronitt Rubinfield. "I've Seen "Enough": Incrementally Improving Visualizations to Support Rapid Decision Making." In: *Proc. VLDB Endowment* 10.11 (2017), pp. 1262–1273. DOI: 10.14778/3137628.3137637.

[98] Vincent Raveneau. "Interaction in Progressive Visual Analytics: Application to Progressive Sequential Pattern Mining." PhD thesis. Université Nantes, Nov. 2020. URL: https://tel.archives-ouvertes.fr/tel-03106201.

[99] René Rosenbaum and Heidrun Schumann. "Progressive refinement: more than a means to overcome limited bandwidth." In: *Proc. of VDA*. Vol. 7243. International Society for Optics and Photonics. SPIE, 2009, pp. 145–156. DOI: 10.1117/12.810501.

[100] René Rosenbaum, Jian Zhi, and Bernd Hamann. "Progressive parallel coordinates." In: *2012 IEEE Pacific Visualization Symposium*. 2012, pp. 25–32. DOI: 10.1109/PacificVis.2012.6183570.

[101] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. "Incremental Learning for Robust Visual Tracking." In: *International Journal of Computer Vision* 77 (2008), pp. 125–141. DOI: 10.1007/s11263-007-0075-7.

[102] H. Schulz, M. Angelini, G. Santucci, and H. Schumann. "An Enhanced Visualization Process Model for Incremental Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 22.7 (2016), pp. 1830–1842. DOI: 10.1109/TVCG.2015.2462356.

[103] D. Sculley. "Web-Scale k-Means Clustering." In: *Proc. of WWW*. ACM, 2010, pp. 1177–1178. DOI: 10.1145/1772690.1772862.

[104] Burr Settles. *Active Learning Literature Survey*. Tech. rep. Department of Computer Sciences, UW-Madison, 2009. URL: http://digital.library.wisc.edu/1793/60660.

[105] Ben Shneiderman. "Direct Manipulation: A Step beyond Programming Languages." In: *Proc. of CHI*. ACM, 1981, p. 143. DOI: 10.1145/800276.810991.

[106] Ben Shneiderman. "Response Time and Display Rate in Human Performance with Computers." In: *ACM Computing Surveys* 16.3 (1984), pp. 265–285. DOI: 10.1145/2514.2517.

[107] C. D. Stolper, A. Perer, and D. Gotz. "Progressive Visual Analytics: User-Driven Visual Exploration of In-Progress Analytics." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 1653–1662. ISSN: 2160-9306. DOI: 10.1109/TVCG.2014.2346574.

[108] Christian Tominski and Heidrun Schumann. *Interactive Visual Data Analysis*. AK Peters Visualization. CRC Press, Apr. 2020. ISBN: 9780367898755.

[109] Edward R. Tufte. *The Visual Display of Quantitative Information (2nd edition)*. Graphics Press, 2007. ISBN: 978-1930824133.

[110] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser. "Designing Progressive and Interactive Analytics Processes for High-Dimensional Data Analysis." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 131–140. DOI: 10.1109/TVCG.2016.2598470.

[111] Stéfan Van Der Walt, S. Chris Colbert, and Gaël Varoquaux. "The NumPy array: A structure for efficient numerical computation." In: *Computing in Science and Engineering* 13.2 (2011), pp. 22–30. ISSN: 15219615. DOI: 10.1109/MCSE.2011.37.

[112] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[113] Jeffrey S. Vitter. "Random Sampling with a Reservoir." In: *Transactions on Mathematical Software* 11.1 (Mar. 1985), pp. 37–57. ISSN: 0098-3500. DOI: 10.1145/3147.3165.

[114] Emily Wall, Leslie M. Blaha, Celeste Lyn Paul, Kristin Cook, and Alex Endert. "Four Perspectives on Human Bias in Visual Analytics." In: *Cognitive Biases in Visualizations*. Ed. by Geoffrey Ellis. Cham: Springer, 2018, pp. 29–42. DOI: 10.1007/978-3-319-95831-6_3.

[115] Guizhen Wang, Jingjing Guo, Mingjie Tang, José Florencio de Queiroz Neto, Calvin Yau, Anas Daghistani, Morteza Karimzadeh, Walid G. Aref, and David S. Ebert. "STULL: Unbiased Online Sampling for Visual Exploration of Large Spatiotemporal Data." In: *Proc. of VAST*. 2020, pp. 72–83. DOI: 10.1109/VAST50239.2020.00012.

[116] Colin Ware. *Information Visualization: Perception for Design (3rd edition)*. Morgan Kaufmann, 2012. ISBN: 978-0-12-381464-7.

[117] Ethan Waterink, Jiri Kosinka, and Steffen Frey. "Visual Analysis of Popping in Progressive Visualization." In: *Proc. of STAG*. Eurographics Association, 2021, pp. 151–162. DOI: 10.2312/stag.20211485.

[118] M. Williams and T. Munzner. "Steerable, Progressive Multidimensional Scaling." In: *Proc. of VIS*. IEEE, 2004, pp. 57–64. DOI: 10.1109/INFVIS.2004.60.

[119] Helen Wright, Robin H. Crompton, Sanjay R. Kharche, and Petra Wenisch. "Steering and visualization: Enabling technologies for computational science." In: *Future Generation Computer Systems* 26.3 (2010), pp. 506–513. DOI: 10.1016/j.future.2008.06.015.

[120] Y. Wu, N. Cao, D. Archambault, Q. Shen, H. Qu, and W. Cui. "Evaluation of Graph Sampling: A Visualization Perspective." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 401–410. DOI: 10.1109/TVCG.2016.2598867.

[121] Peng Xie, Wenyuan Tao, Jie Li, Wentao Huang, and Siming Chen. "Exploring Multi-dimensional Data via Subset Embedding." In: *Computer Graphics Forum* 40.3 (2021), pp. 75–86. DOI: 10.1111/cgf.14290.

[122] R. Xu and D. Wunsch. "Survey of clustering algorithms." In: *IEEE Transactions on Neural Networks* 16.3 (2005), pp. 645–678. DOI: 10.1109/TNN.2005.845141.

[123] J. Yang and J. Widom. "Incremental computation and maintenance of temporal aggregates." In: 12 (2003), pp. 262–283. DOI: 10.1007/s00778-003-0107-z.

[124] S. Young, I. Arel, T. P. Karnowski, and D. Rose. "A Fast and Stable Incremental Clustering Algorithm." In: *Proc. of ITNG*. IEEE, 2010, pp. 204–209. DOI: 10.1109/ITNG.2010.148.

[125] Lingyun Yu, Konstantinos Efstathiou, Petra Isenberg, and Tobias Isenberg. "CAST: Effective and Efficient User Interaction for Context-Aware Selection in 3D Particle Clouds." In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 886–895. DOI: 10.1109/TVCG.2015.2467202.

[126] Jun Yuan, Shouxing Xiang, Jiazhi Xia, Lingyun Yu, and Shixia Liu. "Evaluation of Sampling Methods for Scatterplots." In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 1720–1730. DOI: 10.1109/TVCG.2020.3030432.

[127] E. Zgraggen, A. Galakatos, A. Crotty, J. Fekete, and T. Kraska. "How Progressive Visualizations Affect Exploratory Analysis." In: *IEEE Transactions on Visualization and Computer Graphics* 23.8 (2017), pp. 1977–1987. DOI: 10.1109/TVCG.2016.2607714.

[128] Ji Zhang. "Advancements of Outlier Detection: A Survey." In: *EAI Endorsed Transactions on Scalable Information Systems* 1.1 (Feb. 2013), e2:1–e2:26. DOI: 10.4108/trans.sis.2013.01-03.e2.

[129] Y. Zheng, Y. Ou, A. Lex, and J. M. Phillips. "Visualization of Big Spatial Data using Coresets for Kernel Density Estimates." In: *Proc. of VDS*. 2017, pp. 23–30. DOI: 10.1109/VDS.2017.8573446.

[130] Yan Zheng, Jeffrey Jestes, Jeff M. Phillips, and Feifei Li. "Quality and efficiency for kernel density estimates in large data." In: *Proc. of SIGMOD*. ACM, 2013, pp. 433–444. DOI: 10.1145/2463676.2465319.

[131] Liang Zhou, Chris R. Johnson, and Daniel Weiskopf. "Data-Driven Space-Filling Curves." In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 1591–1600. DOI: 10.1109/TVCG.2020.3030473.

[132] Z. Zhou, C. Shi, X. Shen, L. Cai, H. Wang, Y. Liu, Y. Zhao, and W. Chen. "Context-aware Sampling of Large Networks via Graph Representation Learning." In: *IEEE Transactions on Visualization and Computer Graphics* (2020), pp. 1–10. DOI: 10.1109/TVCG.2020.3030440.

[133] Zhiguang Zhou, Xinlong Zhang, Zhendong Yang, Yuanyuan Chen, Yuhua Liu, Jin Wen, Binjie Chen, Ying Zhao, and Wei Chen. "Visual Abstraction of Geographical Point Data with Spatial Autocorrelations." In: *Proc. of VAST*. IEEE, 2020, pp. 60–71. DOI: 10.1109/VAST50239.2020.00011.

[134] F. van Ham and A. Perer. ""Search, Show Context, Expand on Demand": Supporting Large Graph Exploration with Degree-of-Interest." In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 953–960. DOI: 10.1109/TVCG.2009.108.

[135] F. van Ham and B. Rogowitz. "Perceptual Organization in User-Generated Graph Layouts." In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1333–1339. ISSN: 2160-9306. DOI: 10.1109/TVCG.2008.155.