ABSTRACT

Title of dissertation: TOWARDS EFFICIENT PRESENTATION AND INTERACTION IN VISUAL DATA ANALYSIS

Zhe Cui
Doctor of Philosophy, 2019

Dissertation directed by: Professor Niklas Elmqvist
College of Information Studies
and Professor Joseph JaJa
Department of Electrical and Computer Engineering

The "data explosion" since the era of the Internet has increased data size tremendously, from several hundred Megabytes to millions of Terabytes. Large amounts of data may not fit into memory, and a proper way of handling and processing the data is necessary. Besides, analyses of such large scale data requires complex and time consuming algorithms. On the other hand, humans play an important role in steering and driving the data analysis, while there are often times when people have a hard time getting an overview of the data or knowing which analysis to run. Sometimes they may not even know where to start. There is a huge gap between the data and understanding.

An intuitive way to facilitate data analysis is to visualize it. Visualization is understandable and illustrative, while using it to support fast and rapid data exploration of large scale datasets has been a challenge for a long time. In this dissertation, we aim to facilitate efficient visual data exploration of large scale datasets

from two perspectives: efficiency and interaction. The former indicates how users could understand the data efficiently, this depends on various factors, such as how fast data is processed and how data is presented, while the latter focuses more on the users: how they deal with the data and why they interact with the system in a particular way.

In order to improve the efficiency of data exploration, we have looked into two steps in the visualization pipeline: rendering and processing (computations). We first address visualization rendering of large dataset through a thorough evaluation of web-based visualization performance. We evaluate and understand the page loading effects of Scalable Vector Graphics (SVG), a popular image format for interactive visualization on the web browsers. To understand the scalability of individual elements in SVG based visualization, we conduct performance tests on different types of charts, in different phases of rendering process. From the results, we have figured out optimization techniques and guidelines to achieve better performance when rendering SVG visualization.

Secondly, we present a pure browser based distributed computing framework (VisHive) that exploits computational power from co-located idle devices for visualization. The VisHive framework speeds up web-based visualization, which is originally designed for single computer and cannot make use of additional computational resources on the client side. It takes advantage of multiple devices that today's users often have access to. VisHive constructs visualization applications that can transparently connect multiple devices into an ad-hoc cluster for local computation. It requires no specific software to be downloaded for setup.

To achieve a more interactive data analysis process, we first propose a proactive visual analytics system (DataSite) that enable users to analyze the data smoothly with a list of pre-defined algorithms. DataSite provides results through selecting and executing computations using automatic server-side computation. It utilizes computational resources exhaustively during data analysis to reduce the burden of human thinking. Analyzing results identified by these background processes are surfaced as status updates in a feed on the front-end, akin to posts in a social media feed. DataSite effectively turns data analysis into a conversation between the user and the computer, thereby reducing the cognitive load and domain knowledge requirements on users.

Next we apply the concept of proactive data analysis to genomic data, and explore how to improve data analysis through adaptive computations in bioinformatics domain. We build Epiviz Feed, a web application that supports proactive visual and statistical analysis of genomic data. It addresses common and popular biological questions that may be asked by the analyst, and shortens the time of processing and analyzing the data with automatic computations.

We further present a computational steering mechanism for visual analytics that prioritizes computations performed on the dataset leveraging the analyst's navigational behavior in the data. The web-based system, called Sherpa, provides computational modules for genomic data analysis, where independent algorithms calculate test statistics relevant to biological inferences about gene regulation in various tumor types and their corresponding normal tissues.

TOWARDS EFFICIENT PRESENTATION AND INTERACTION IN VISUAL DATA ANALYSIS

by

Zhe Cui

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever, bitter and sweet.

First and foremost I'd like to thank my research advisor, Professor Niklas Elmqvist for his continuous support through my Ph.D study. I still remember the first time when I came to his office and talked with him about the potential research opportunities. I am extremely grateful for his patience, motivation, and foresight on the challenging problems we solved together. He has always guided me through the dark cloud of research and has faith in me during those difficult times. It has been a pleasure to work with and learn from such an extraordinary and considerate researcher, husband, and father.

I would like to thank my academic advisor, Professor Joseph JaJa. Without his guidance in my research and flexibility in administrative issues, this thesis and even my Ph.D study would have been a distant dream. He is always very nice, helpful, and responsible.

I would like to thank Professor Héctor Corrada Bravo for introducing me to bioinformatics, in which we figured out a very good application domain of the thesis work. His expertise and kindness inspired me to boldly move forward without hesitating.

I would also thank Professor Ashok Agrawala, who was my mentor during my intial time at UMD. He inspired and encouraged me a lot and were always willing to help. I am really grateful for that. Thanks also goes to Professor Gang Qu and Professor Donald Yeung for agreeing to serve on my thesis proposal and defense committee and for sparing their invaluable time reviewing the thesis and providing feedback.

I would like to extend my gratitude to all my colleagues and collaborators in Professor Elmqvist, Professor JaJa, and Professor Bravo's team. First thanks go to Karthik Badam, Adil Yalcin, Jayaram Kancherla, and Senthil Chandrasegaran, for your continuous support and guidance to help me step into the field, for invaluable suggestions and inspiring discussions. Thanks go to Zhenpeng Zhao, a great colleague and friend, for those enjoyable discussions, guidance, and all the fun we had. Thanks also go to other colleagues, Deok Gun Park, Matthias Nielson, and Andrea Batch for all the time we spent together. You have enriched my life at University of Maryland in many ways.

I would also like to acknowledge help and support from staff members in ECE department, Melanie Prange, Emily Irwin, and Maria Hoo. You helped with a lot of administrative issues and paperwork, which made my experience at UMD smooth and enjoyable.

I owe my deepest thanks to my family, my mother and father who have always stood by me and guided me all the way today. Thanks also go to my dear wife, Biying Li, for her endless love, understanding, and support throughout the studies. They always cared about me, encouraged me, and accompanied me throughout those ups and downs no matter what happened, not only for the graduate study, but for my life. Words cannot express the gratitude I owe them.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Big data is large volumes of high velocity, complex and variable data [1,2] that requires advanced techniques to manage and process. People analyze *big data* and get significant value and benefits from it in many different fields, such as finance, medicine, and transportation. This comes to *Big data analytics* [3], which examines large amounts of data to uncover hidden patterns, correlations and other insights. Nowadays, big data analytics require both complex processing as well as human understanding to achieve easy interpretation and decision making. As a result, how to manage available resources to support rapid processing of data, and alleviate human thinking load is essential to the success of big data analytics.

Although there are many ways to tackle big data problems, visualization is the one that is easy to come up with. *Visualization* is the technique that creates visual representations (e.g., images, diagrams, and animations [1]) to convey message or information. *Data visualization* is any effort or technique that combines data in visual context to help understand the data. It has been very popular in almost all domains: stock market, daily navigation, and dish recipes. The widely use is also reflected in a variety of visualization tools. There has also been a rapid growth of *vi-*

[1]From Wikipedia on Visualization `https://en.wikipedia.org/wiki/Visualization_(graphics)`

sual analytics which focuses on "analytical reasoning facilitated by interactive visual interfaces" [4, 5] in the field of data visualization or in a broader sense, information visualization. At one end visualization and visual data analytics [6, 7] is not primarily designed for large scale datasets, it is difficult to simply apply traditional visual analytics system directly. At the other end humans are more and more involved in the analytical process of big data, how to embed human interactions and behavior into the workflow is crucial to the analysis. Thus, it is necessary to achieve efficient data exploration via a combination of computational resources, interactions, and sensemaking.

1.1 Challenges in Visualization for Big Data

As per the definition of visualization and visual analytics (VA) [8], many VA applications require significant computations—such as clustering [9], word embedding [10], and inferential statistics—to be run on new datasets prior to presentation to the user. On the other hand, web browser becomes one of the most popular platform for modern visualization. The web has a lot to offer visualization developers, such as advanced accelerated graphics and integration with the entire web ecosystem, including remote databases, sophisticated web services, and online geographical map systems. More importantly, web browser is now ubiquitous on all devices—from laptop to smartphone, tablet to smartwatch. It is very easy and simple to acquire information on the browser from various sources. However, real-world datasets are increasingly reaching a volume and complexity where such computation can be forbiddingly costly in terms of computation and time. The browser is not an ideal computational environment for executing complex algorithms that many visualization applications require.

With regard to this, one major research area is high performance visualization [11], which targets to achieve faster data processing and better visualization rendering. High performance visualization aims to utilize computer resources as much as possible and at the same time, optimize performance on the browser. While web is not primarily designed for heavy loaded computation, people nowadays have multiple devices in possession but they mostly use one at a time to stay focused. The potentials to leverage computational resources of idle devices with simple setup can be a major advantage.

1.2 Human Factors in Visual Analytics

Another challenge comes from human. Data exploration using visual analytics is often characterized as a partnership between the analyst and computer, with each partner providing unique and complementary capabilities [8, 12].

Most visual analytics systems have long been running in a passive mode and put the analyst in the driver's seat to guide the analysis, i.e., waiting for the inputs from the analyst and executes whatever requested. This kind of analysis heavily depends on cognitions of the analysts and actions he/she takes, and it falls short when the analyst does not know how to best transform or visualize the data, or is simply overwhelmed due to the sheer scale of the dataset or limited time available for analysis.

Another type of visual analytics systems would share control between the two sides—analyst and computer—in a way that leverages their respective strengths. This *Computer-as-partner* technique, as opposed to *Computer-as-tool* [13], would automatically select and execute appropriate computations to inform the analyst's exploration and sensemaking process. Specifically, when exploring, the user analyzes and visualizes the data which does need a lot of CPU resources while a computation engine simultaneously runs analyses in the background. The underlying design rationale is that CPU cycles are cheap, whereas human cognitive effort is not, and while computational resources are idle during the user exploration process, the system utilizes these resources in some way to aid user's understanding procedure.

1.3 Boosting Visual Data Analysis

As stated above, the research field of combining available algorithmic computations with efficient visual analysis to aid the process of data exploration has a lot to unfold. Specifically, we enumerate different perspectives that drive and facilitate visual data exploration:

1.3.1 Computational Resources

nowadays, more than 4 billion mobile phones are used in the world [14]. Also, people have multiple devices in hand, such as laptops, smartphones, tablets, while most of them are idle most of the time. On the other hand, visual data exploration requires faster response and lower interaction latencies, which demand a lot of CPU resources. The imbalance situation pushes us to take advantage of idle devices to lower the resource gap between expected requirements for visual analysis and existing resources on various devices.

1.3.2 Mitigating "Cold Start" for Analysis

The "Cold Start" [15] is a prevalent problem in recommender system. Similarly, the challenge during data exploration is: when the analyst starts analyzing a dataset, it is difficult and sometimes impossible to quickly get immediate results because the dataset can be overwhelming and difficult to handle. To mitigate the situation, analyses can be performed using available computational resources *automatically* to provide relevant results to the user during the exploration.

1.3.3 User Interactions

User interactions/behaviors are an important part of visual analysis [16], especially for the "computer-as-partner" paradigm [13], where user communicates with the computer and pushes the analysis forward. While existing interaction based analysis refinement mostly depends on user's activities, the need for integrating automatic computations into the data exploration procedure is a necessity. For example, when people analyze gene expression data, they zoom into a region within a chromosome, the computer can prioritize available resources to execute statistical analysis results within that region since it's likely they are interested in that gene sequence. The earlier they see region specific results, the faster they can understand the sequence.

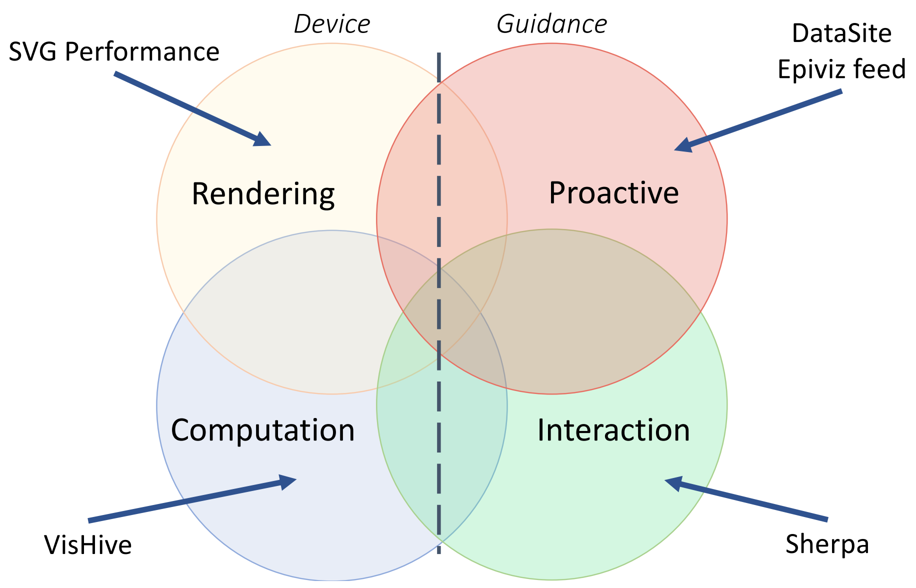


Figure 1.1: Structure of the dissertation in visual data analysis and the corresponding components.

1.4 Thesis Outline

In this thesis, we first provide in-depth background of visual analytics and recommendations in Chapter 2. This includes visualization performance evaluation, distributed computing across mobile devices, visualization scalability, and recommender systems for visualization. To address each of the above scenarios, the main thesis work is introduced in Chapter 3 to 7.

In Chapter 3, we evaluate the web-based visualization performance to get a firm understanding of the influencing factors for rendering time and latencies of Scalable Vector Graphics (SVG), as well as the techniques to improve it.

In Chapter 4, we introduce a distributed computing framework for web browsers (called “VisHive”), which leverage computational resources of idle devices to create ad-hoc clusters that manages computing tasks for visualization communicated using web browsers, with only a matchmaking service and no client side setup.

To address “Cold Start” problem in visual analytics, in Chapter 5 we present a proactive visual analytics system (called “DataSite”), where the user analyzes and visualizes the data while a computation engine simultaneously selects and executes appropriate automatic analyses on the data in the background. We evaluate the approach both qualitatively and quantitatively with a comprehensive user study. We further apply the framework in bioinformatics domain, and build an application that works for genomic data (“Epiviz Feed”).

In Chapter 7, we present a computational steering mechanism (called “Sherpa”) for progressive visual analytics that automatically prioritizes computations performed on the dataset based on the analyst’s navigational behavior in the data. We conducted expert reviews with genomic and visualization experts, and found that Sherpa provided comparable accuracy and shorter analytical time compared to computations without priority management. The overall structure of the thesis is shown in Fig. 1.1.

Finally, we conclude the thesis and propose future work in Chapter 8.

Chapter 2: Background

This thesis builds upon a body of research and practice on big data visualization, distributed computing, visualization recommendation, and progressive visual analytics. In this section, we discuss existing literature in each field and how this thesis extends them.

2.1 Visualization on the Web

Visualization has been around for a couple of decades, while in the early 1990s, data visualization was still considered an emerging discipline. Towards the turn of the century, however, the pervasiveness of the web had led to many changes, including one important application: visualization in the browser. Rohrer et al. [17] note that the web is essentially a fundamentally new medium for visualization. Today, virtually all computational devices—both computers and mobile devices—provide full-fledged web browsers as part of their standard software distributions.

Web-based visualization toolkits include Protovis [18] and D3 [19] as well as more generic graphics toolkits such as Processing.js, Raphaël, and Paper.js. Most prominent of these is D3, proposed by Bostock et al. [19], which provides a direct binding between the input data and the document object model.

Targeting the web platform also implies dealing with the restricted computing and rendering abilities of modern web browsers. Meanwhile, work in distributed computing is trying to achieve the same success by using the browser and the web as the base platform for parallel and high-performance computing. Martinez and Val [20] first proposed the idea of using standard web technologies for distributed computation across multiple devices in 2014, and later presented the Capataz [21] framework for distributed algorithms across the web. While Capataz is not designed for visualization and has a server/client architecture, we strive to develop a peer-to-peer system without specific computational server, which is more convenient to use.

2.2 Big Data Visualization

While combining big data and visualization introduces a lot of opportunities, it also comes with many challenges. When dealing with large scale datasets, the bottleneck could come from either rendering or data management.

2.2.1 Visualizing Large Scale Datasets

Visualizing big datasets on conventional displays can lead to overplotting, which overwhelms the user's perceptual and cognitive capacities [22]. Data reduction methods such as sampling [23, 24], aggregation [25–27], and filtering [28] have therefore been proposed to support perceptual scalability. More sophisticated versions of these techniques have also been proposed including kernel density estimation methods for specific visualization types [29] and hierarchical aggregation [30] to transform any visualization into a multiscale visual structure.

Big data visualization typically involves two main challenges: perceptual and computational scalability [30]. Representative work of perceptual scalability includes that of Ahlberg and Shneiderman [28] for filtering, Das Sarma et al. [31] for spatial sampling, and Carr et al. [32] for aggregation of scatterplots. We will not discuss further since this is not the focus of the thesis. The techniques used for big data visualization also depend strongly on data type. For example, Fisher et al. [33] show techniques for tackling business intelligence, Wong et al. [34] discuss challenges facing extreme-scale visual analytics, and Steed et al. [35] developed a visual analytics system for the analysis of complex earth simulation datasets.

Recent years have seen an influx of work on computational scalability for visualization. Liu et al. [22] developed a visual analysis system called imMens, which uses WebGL for data processing and is based on the principle that scalability should be limited by the chosen resolution of the visualized data and not the total number of records. Nanocubes [36] is another approach focused on visualizing and analyzing very large datasets based on a compact data cube representation. Choo and Park [37] propose methods such as data scale confinement, classification of pre-clustered data, and linear transforms of higher dimensions to deal with scalability for visualization. Finally, a recent trend in tackling big data for visualization is *progressive visual analytics* (PVA) [38], where partial results from complex and lengthy computations are visualized during the process, allowing the user to better guide the analysis. We will discuss PVA later in this chapter.

2.2.2 Databases and Visualization

Data visualization targets to make data analysis easy to understand, and databases are the fundamental data sources. Polaris [39], VisDB [40], VQE [41] all focused on developing visualization techniques that directly support interactive multi-dimensional database exploration through visual queries. In terms of huge databases that is impossible to construct queries and get immediate response in runtime, there are pre-fetching and pre-computation techniques [22, 42–44] to support database queries, as well as speed up executions. Users can utilize these tools to construct queries directly through their interactions with the interface. These techniques map query results to visualizations.

2.3 Distributed Computing on Mobile Devices

Distributed computing and systems have long been extensively studied [45]. It is well applied in modern computer age and programming language field. When it comes to distributed computing on mobile devices, Lin et al. [46] first proposed a mobile network where nodes would be organized into non-overlapping clusters that are independently controlled and dynamically loaded. The proposed cluster algorithm is robust to node failure or addition/deletion. Wang et al. [47] presented a bandwidth adaptive clustering approach for mobile ad-hoc networks that maintains clusters using local topology information only. In their approach, the member nodes forward only the maintenance messages probabilistically based on available bandwidth. This ensures adaptability to network conditions and reduces message overhead. Lee et al. [48] discussed the challenges and advantages of utilizing mobile devices for distributed analytics based on an implementation of the Hadoop framework. Based on a performance analysis of their implementation, they concluded that current mobile devices face significant limitations on transmitting and receiving reliable TCP data streams, which is required to avoid interruptions during distributed analytics.

A number of computation offloading frameworks have been proposed for computationally intensive mobile applications [49–51]. Such applications are said to be *elastic* in nature, and each approach partitions problems at different levels of granularity at runtime. In most cases, the distributed application processing platform is composed of a mobile device that runs a local application, a wireless network medium, and a remote cloud server node. In cases where there are insufficient resources on the mobile device, an elastic mobile application can be partitioned such that any computationally intensive components of the application can be offloaded during runtime. Hassan et al. [52] showed in their study of computing-intensive mobile applications that outsourcing these computations to nearby residential computers or devices may be more advantageous than public clouds due to network impact. Cuckoo, a computation offloading framework for smartphones developed by Kemp et al. [53], allows computation offloading for Android phones to a remote server. Shiraz et al. [51] showed that current mobile computational offloading frameworks implement resource-intensive procedures for offloading. This involves the overhead of transmitting application binary code as well as deploying distributed platforms at runtime. Runtime computational offloading is also useful in decentralized distributed platforms, such as mobile ad-hoc networks. Shiraz et al. note, however, that remote server nodes are unpredictable and computational offloading should therefore be performed on an ad-hoc basis at runtime. This motivated us to design our framework as an ad-hoc network of mobile devices that perform computations on demand.

2.4 Exploratory Visual Analysis

Exploratory data analysis (EDA) [6,54] is the canonical user scenario for visualization. The key characteristic for EDA is that the analyst is not initially familiar with the dataset, and may also be unclear about the goals of the analysis. The exploratory process involves interactively browsing the data to get an overall understanding, deriving questions from the data, and finally looking for answers.

Efficient data exploration often relies on visual interfaces [6]. *Dynamic queries* [55] is an interaction technique for such interfaces, where users formulate visual queries as a combination of filters. Writ large, *faceted browsing* allows for creating queries on specific dimensions of the data [56].

2.5 Visualization Recommendation

The idea behind visualization recommendation is to use recommendation engines [57] to suggest relevant views to the user, thus reducing the cognitive load. While this idea has seen a resurgence in the visualization community in recent years, it is by no means a new idea. Mackinlay [58] first proposed automatic visualization design based on input data in 1986. His work combines expressiveness and effectiveness criteria inspired by Bertin [59] and Cleveland et al. [60] to recommend suitable visualizations. In 2007, Tableau's Show Me system [61] finally provided a practical and commercial implementation of these ideas.

Many similar approaches to automatic visual specification exist. Sage [62] extends Mackinlay's work to enhance user-directed design by completing and retrieving partial specifications based on their appearance and data contents. The rank-by-feature framework [63] sorts scatterplot, boxplots, and histograms in a hierarchical clustering explorer to understand and find important features in multidimensional datasets. SeeDB [64] generates a wide range of visualizations, and define which ones would be interesting by deviation and scale. Perry's [65] and Van den Elzen's [66] work attack the problem that generates multiple visualizations shown with small thumbnails.

Recommendation engines have been used to great effect for visualization in the last few years. Voyager [67] generates a large number of visualizations and organizes them by relevance on a large, scrolling canvas. Visualization by demonstration [68] lets the user demonstrate incremental changes to a visualization, and then gives recommendations on transformations. Zenvisage [69] automatically identifies and recommends interesting visualizations to the user depending on what they are looking for. Finally, Voyager 2 [70] builds on Voyager, but supports wildcards in the specification and provides additional partial view suggestions. All of these ideas were formative in part of this dissertation, but our approach takes this a step further by focusing on continuous computation from a library of automatic algorithms, with findings propagated to the user in a dynamically updating feed, while also involving user's interactions into the loop.

2.6 Computations in Visualization

The process of data analysis includes extraction, preprocessing, filtering, analyzing, transformation, and presenting the results. Many of the steps require algorithmic operations on the dataset. As in many existing visualization tools, the user has to choose what computations need to be run to get the desired visualization. This increases the difficulty in analyzing the datasets efficiently. To remedy this, researchers have worked on automatic analysis as well as computational steering.

2.6.1 Proactive Computation alongside Visualization

The idea of proactive visual analytics discussed in the dissertation builds on the idea to opportunistically run computations in anticipation of user needs, which is observed in Novias [71], Treeversity [72], and Analyza [73] (*Explore* in Google Sheet). Novias identifies visual elements of evolving features and provides multiple views in an interactive environment. Treeversity provides a list of outliers in textual form, which identifies changes in the data automatically. The most similar research to the work in this thesis is Analyza, which provides auto-computed features in natural language. In contrast, the proposed framework (DataSite) aims to push proactive computation to depth and complexity rather than just simple statistics from the overall dataset. Furthermore, DataSite pushes features to a feed view that is akin to social media feeds users are already accustomed to.

2.6.2 Computational Steering

Many computational algorithms, particularly for scientific and simulation purposes, are extremely resource-intensive and time-consuming to complete, often requiring massive computational clusters. For this reason, the notion of *computational steering*—interactive control over a computation during execution [74]—is very attractive, as it allows the scientist or engineer to guide the process in real-time in order to faster converge on a desirable solution. Mulder et al. [75] enumerate uses of computational steering as model exploration, algorithm experimentation, and performance optimization. Examples of well-known computational steering environments include SCIRun [76], Progress/Magellan [77, 78], and VASE [79]; some applications include fluid dynamics (CFD) [80], program and resource steering [77], and high performance computing (HPC) platforms [81].

Most computational steering mechanisms are *explicit*, in that they give the user direct control over the ongoing computation using operations that are specific to the domain. However, this may require significant expertise on behalf of the user. Recent efforts have focused on coupling interactive visualization with computational steering to display intermediate results as well as provide more straightforward controls. World Lines [82], Nodes on Ropes [83], and Visdom [84] are all examples of such integrated steering environments, typically used to control multiple runs of the same or related simulation models with slightly perturbed inputs. Similarly, VASA [85] is a visual analytics system for asynchronous computational steering of large simulation pipelines. Overall, computational steering can help computations converge on the appropriate results faster, but implicit steering that provides interactive visual representations for incremental results will help reduce the user's required expertise.

2.7 Progressive Visual Analytics

The tremendous leap in computational power over the last few decades has so far mostly benefited confirmatory analysis, where the analyst initializes a model and then executes it, waits minutes, hours, and sometimes days for the computation to finish. A more exploratory data analysis [6], such as those supported by interactive visualization and analytics [8], requires a tightly optimized feedback loop with latency of at most 10 seconds or less (often around 0.5 seconds [86]). To make big data analytics [87] responsive in such interactive and exploratory settings, recent work has proposed the concept of *progressive visual analytics* (PVA) [88, 89], where intermediate results are continuously fed back to the visualization to show gradual progress over time.

While PVA nominally includes computational steering as one of its main components [89], few practical implementations provide steering capabilities. The original ProgressiVis Python toolkit [90] has "optional input slots," but these are never explained in detail. Zgraggen et al. [89] evaluate PVA for three output conditions, including blocking, instantaneous, and progressive, but does not involve the input side—i.e., user-controlled steering—in their study. PANENE [91] is a progressive tree structure for nearest neighbor computations, but does not expose steering controls to users.

In contrast, Badam et al. [88] explore user interfaces for PVA in particular, providing process controls—pause, stop, and progress bars—as well as algorithm-specific options for controlling the ongoing execution. However, the process controls are simplistic, whereas the algorithm options merely expose the raw parameters of the computation, and thus require some expertise to manipulate. The incremental query visualizations proposed by Fisher et al. [33] provide similar basic controls for pausing, resuming, and canceling an ongoing query. Finally, a recent progressive implementation of t-SNE dimensionality reduction allows the user to control which part of the data to focus on first [92]. This approach is highly relevant to our approach in that it provides a user-controlled Magic Lens [93] that will also steer the computation. However, the approach is specific to t-SNE embedding, and puts less focus on the navigational behavior. Most current PVA systems focus on iterative updates of the visual representation and less on computational steering controls.

Chapter 3: Performance Evaluation of Scalable Vector Graphics for Web-Based Visualization

Vector graphics has become increasingly popular in recent years, particularly in the form of Scalable Vector Graphics (SVG) format [94, 95]. As the support for SVG in modern web browsers has expanded, third-party JavaScript libraries—such as D3 [19] and Vega [96]—that leverage vector graphics to create interactive visualizations have emerged. However, web-based visualizations tend to perform poorly when designers encode large datasets into vector graphics in a straightforward manner, which often results in complex SVG graphics with a large number of elements. This may yield high rendering time and high latency, causing unresponsive interaction and resulting in poor user experience. In order to help visualization designers make informed decisions on how to handle such large datasets while retaining responsiveness in scalable vector-based visualization, a firm understanding of the influencing factors as well as possible techniques for addressing them is required. This makes managing large datasets in the web browser a potent and largely unaddressed challenge for developers, who often rely on rules of thumb for how many elements can be rendered while retaining interactivity and performance.

Several general techniques for improving rendering performance and reducing latency of SVG exists, such as aggregation [30], sampling [97], and progressive rendering [90]. While such techniques are indeed useful, they do not address the underlying rendering factors that prompt such techniques. To study these underlying factors we conduct an in-depth investigation of performance in vector graphics generation and rendering of two basic visualization techniques—scatterplots and parallel coordinate plots—representing two-dimensional and multidimensional datasets. We implement and evaluate rendering performance of these visualization techniques in a modern web browser (Google Chrome) using D3 JavaScript library [19]. From these results, we are able to describe detailed relationships between rendering performance on one side, and on the other side the number of visual elements, CSS styling properties, visualization size, visual element size, and visual element decimal precision. We leverage these findings to provide a set of practical guidelines to help improve rendering performance of web-based SVG visualizations.

3.1 Framework: Web Visualization

We investigate techniques for handling and creating visualizations of large datasets, and base our conclusions on thorough empirical studies of browser rendering performance of SVG visualizations. The process of transforming raw data into visual elements expects a structured form of data storage to associate items in a dataset with graphical primitives. However, many modern domains deal with raw data (e.g., text), which requires significant pre-processing to convert into relational data tables with hierarchies. Beyond this, the visualization pipeline itself expects data transformations between subsequent stages [98]. Examples include (1) visualization transformation using algorithms such as Multi-Dimensional Scaling (MDS), and (2) mapping transformation for layout computation (e.g., force-directed layout). In web visualization, systems such as iMmens [22] have processed large amounts on the web browser using parallel architectures. Furthermore, the choice of transformation algorithms depends on the visualization design, and can often be performed offline. Therefore, we assume availability of processed structured data on the browser cache for JavaScript code to create visual representations, and evaluate rendering performance for SVG visualizations.

3.2 Performance Evaluation

In this section, we outline our test methodology and report in detail on the performance of visualizing datasets using SVG in a browser. In our experiment, we measure these time periods for rendering conventional implementations of scatterplots and parallel coordinate plots, both implemented using D3 [19]. We have chosen these two visualization techniques because they are both *unit visualizations*, i.e., they visualize one data point with one visual mark, and when they are used to visualize thousands or even tens of thousands of visual marks, their rendering performance can struggle. We measure DOM manipulation time, style calculation time, and pixel rendering time, and based on these measurements, we are able to demonstrate detailed relationships between numbers of elements, styling properties, and properties of visual marks. We then outline how to achieve performance gains by omitting certain CSS styles or reducing the decimal precision of visual elements. Furthermore, we present three techniques including sampling, aggregation, and progressive rendering, that ensure responsiveness when rendering SVG visualizations. Our results provide an in-depth understanding of what factors are at play when rendering SVG visualizations in web browsers, and thus enable visualization developers to make informed decisions of how to achieve desired responsiveness.

3.2.1 Method and Apparatus

We focus on controlled experiments, where we perform automated tests by changing multiple factors (one at a time). For both visualizations we vary the number of visual marks, the dimensions (area) of the visualization, the styles applied to the marks, and the coordinate precision of the marks. Furthermore, when reporting on the three techniques mentioned above, we measure performance for progressively rendering both scatterplot and parallel coordinate plots.

To test responsiveness on interaction with SVG visualizations in a browser, we make a *worst-case assumption* based on the deliberation that the worst-case result of an interaction is that the entire visualization must be redrawn, meaning a user must wait until the redrawing is complete. Therefore, we rigorously measure the time it takes to render a SVG visualization from data over DOM manipulation and styling to rendered visualization, but understand this as the worst-case consequences of interactions. In another word, we do not measure particular interactions or interaction techniques. Furthermore, we perform measurements of up to 100k ele-

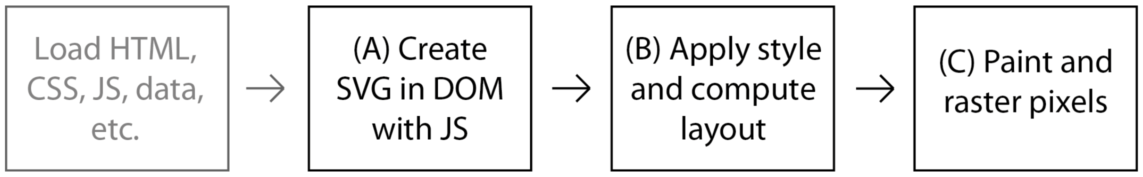

Load HTML, CSS, JS, data, etc.
(A) Create SVG in DOM with JS
(B) Apply style and compute layout
(C) Paint and raster pixels

Figure 3.1: Simplified pipeline of browser workflow.

ments, which might be considered unrealistic or impractical to visualize with SVG in a browser. The dataset we use is a publicly available collection of 20 years of domestic flights in the U.S.

All measurements are performed under the same conditions on a commercial laptop computer with an Intel i7 CPU with a base frequency of 1.6 GHz and turbo frequency of 2.6 GHz, a 14” 2560×1440 resolution monitor, and running the Microsoft Windows 10 operating system. The computer performed only rendering performance measurements. All renderings of visualizations were made in a standard version of Google Chrome version 63 Stable with GPU acceleration enabled. The browser windows always have focus, and all pixels of rendered visualizations were visible inside the browser's viewport without scrolling.

3.2.2 Primer: Browser Rendering Pipeline

When creating a SVG visualization, the browser performs the following operations (see also Fig 3.1): (A) The DOM is manipulated by inserting or manipulating SVG nodes; (B) the nodes in the DOM are styled and the webpage's layout is computed (henceforth style and layout is referred to only as style/styling for brevity), and (C) the visual elements are painted. The exact implementation, and hence performance, will differ from browser to browser but all browsers need to perform these operations. We considered evaluating our implementations in the four major desktop web browsers: Apple Safari, Internet Explorer, Mozilla Firefox, and Google Chrome. However, we excluded Apple Safari and Internet Explorer in this set of evaluation because they are not fully open source, i.e., the inner workings of the browser are not publicly known. Consecutively, we were forced to disregard Mozilla Firefox as well because it proved unstable when handling large number of elements over multiple iterations.

3.2.3 Experiment Factors

In this section we will briefly elaborate on the factors we vary when conducting measurements as well as their standard values. As listed in Table 3.1, the factors are dimension, number of elements (i.e. visual marks), styling, rounding coordinate precision, and hardware acceleration. When appropriate, due to counter-intuitive oddities, we compare measurement results with hardware acceleration enabled and disabled. However, as disabling/enabling hardware acceleration is not a setting that a visualization developer is generally in control of, we only include these sporadically. We use these standard values as a common denominator throughout all reported measurements, i.e., we compare measurements conducted when varying a factor to these standard values. In the following section we will elaborate on our performance measurements in the same order as in Table 3.1, and for each factor we will describe how we change them to conduct our measurements.

3.2.4 Measure

We measure DOM manipulation, styling, and painting time using JavaScript timeouts and by registering a function to listen for a browser event fired when the browser has painted pixels. We make measurements at four selected points: DOM manipulation is measured by noting the timestamp before *measurement 1* and after *measurement 2*, a loop that manipulates the DOM. This works because the browser is single threaded and thus completes its data insertion loop before continuing. Styling time is measured by requesting a function to be executed after a 0 millisecond delay *measurement 3* immediately after the DOM manipulation loop is done. This works because, in JavaScript, the function passed to a timeout is only requested to be executed after the specified delay, but it will be executed at earliest after the browser's main thread becomes available, which is after the DOM manipulation is finished. Finally, we measure the time taken to paint the pixels by asking the browser to notify us when something has been painted onto the screen, where we provide a callback function, which on execution enables us to measure when painting is done (*time stamp 4*). Even though the painting event notifies when something is *first* painted on the screen, in our use case, this means it is triggered when everything is painted, because we paint all elements in a single batch.

3.2.5 Procedure

We conduct measurements with an automated script loaded into testing computer's browser, which saves measurement data into the browser's *LocalStorage* [1] after completing each iteration. The test data reported consist of 100 permutations of factors all conducted with at least 50 iterations each, resulting in a total of more than $5,000$ visualizations rendered. We conduct 50 iterations for each permutation to reduce the fluctuation of our collected data and all measurement numbers reported in the following section is the average time in seconds of 50 iterations for each individual permutation. To further gauge the validity, we calculated averages and standard deviation of the DOM manipulation time, styling computation time, and pixel painting time for each permutation. To evaluate the quality (i.e., consistency) of our measurements, we divide the individual standard deviations with their corresponding averages, to get a metric that tells us in percentage how large the standard deviation is compared to the average. Less than one tenth of our measurements have a quality metric between 10% and 20%, three measurements have a quality metric between 20% and 33%, and a single measurement—painting time of scatterplot with $70,000$ elements—has a quality metric of 84%. This particular permutation also shows deviation in it's styling computation time, so we attribute this to variations in the data collection, which usually occurs when conducting real world empirical studies.

[1] `https://developer.mozilla.org/en-US/docs/Web/API/Storage/LocalStorage`

3.3 Performance Results

In this section, we report the performance results focusing on two visual representations: scatterplots and parallel coordinates. As mentioned in the previous section, we look into the effects of the number of elements, as well as their sizes, shapes, precision, and styles for each visual representation. These results are compared against a baseline setting (called "golden standard") in Table 3.1. We first look into the results for each visual representation and then compare the two types.

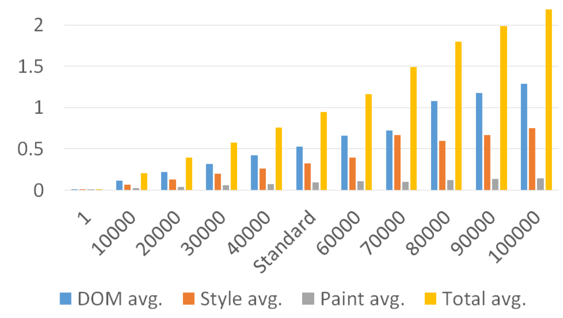
All measurements reported below have been conducted on visualizations of the same size, except for the subsection "Effect of Plot Size" where we explicitly change only the size of the visualizations. Furthermore, all the time periods are measured from 50 iterations, and the standard deviations of the 50 iterations are less than 10% of the mean.

Type	**Elements**	**Wid.**	**Heig.**	**Rd.**	**Styling**
Scatterplot	50000	960	480	False	radius=1 & fill=steelblue & stroke=none
Par. coor.	50000	960	480	False	fill=none & stroke=steelblue

Table 3.1: Baseline parameter settings used in the performance evaluation. Scatterplot and parallel coordinate visual representations have the same settings except the default setting for the radius of the points in scatterplot.

3.3.1 Number of Elements

The first test for each visual representation is measuring the effect of the number of SVG elements being visualized. To this end, we used standard implementations of scatterplots and parallel coordinate plots made with D3. This choice is based on the fact that these unit visualizations [99,100] are both capable of visualizing large number of data items and are prone to overplotting, which is undesirable as it means that a pixel can be painted multiple times. Furthermore, they differ in their visual structures, i.e., the types of shapes and the number of pixels that the web browser needs to paint.



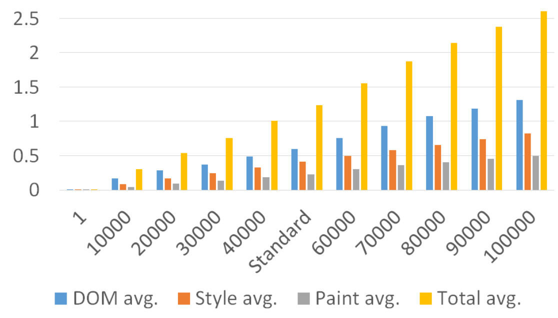


Figure 3.2: Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering scatterplots (left) and parallel coordinates (right) from 1 to 100k elements.

Our measurements show three basic observations: (1) scatterplots are slightly faster to render than parallel coordinate plots; (2) time taken for rendering is linearly proportional to number of elements; and (3) painting time is the primary distinguishing factor in rendering time between scatterplots and parallel coordinate plots. These three patterns are expected and is partially a consequence of overplotting, where elements and individual pixels can be rendered multiple times.

Best Practice 1: Responsive rendering time increases along with the number of elements. To get better performance, render fewer visual elements.

In Fig 3.2, the measurements for rendering the standard scatterplot and parallel coordinate plots are also divided into DOM manipulation time, styling time, and painting time. These figures reveal that the cause of high difference in rendering is mostly due to a longer painting time for parallel coordinate plots. This is because Chrome (and Firefox alike) updates the viewport following a commonly applied "dirty rectangle" principle, where the viewport is divided into rectangles and each element to be painted or repainted marks the rectangles that the element intersects as dirty, triggering the browser to repaint the rectangle. Since a dot in a scatterplot will intersect fewer of the viewport's rectangles than a path in a parallel coordinates visualization, it requires fewer rectangles to be painted.

Besides the difference in painting time, a minor increase in DOM manipulation time and styling time can be noticed. We speculate that this is because a circle in scatterplots is simpler to describe (it requires one x, y coordinate set and a radius) than a path in parallel coordinate plots, which requires a series of x, y coordinate sets. Therefore, the time taken to insert a path node into the DOM is longer than a circle node. However, as noted, the difference in DOM manipulation time between scatterplots and parallel coordinates is minor and thus should be of less concern than the painting time.

Best Practice 2: SVG visualization developers should consider the complexity of visual marks if performance is a requirement. For example, simple visual marks, such as circles or even squates in scatterplots, can enable the visualization to cope with a larger number of nodes and still remain responsive, compared to paths in parallel coordinate plots.

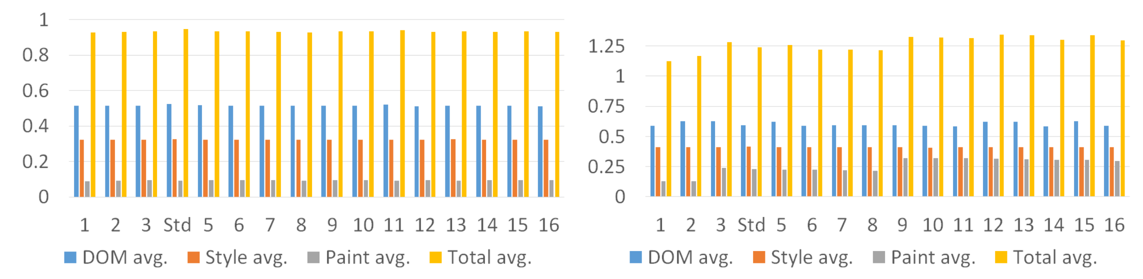


Figure 3.3: Scatterplots (left) and parallel coordinates (right): Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering with linearly increasing area of the visualization. Step 4/Std corresponds to the standard in Table 3.1.

3.3.2 Effect of Plot Size

As seen in Fig. 3.3, the time taken to render a scatterplot stays almost identical for different sizes, while the time to render the parallel coordinate plot increases in a slightly step-wise manner. These trends can be connected back to the "dirty" rendering protocol of the web browser. In case of parallel coordinates, the path will intersect with more "dirty" rectangles as size increases, but the total number of intersected rectangles does not increase at the same rate as the area of the visualization. This is also visible on the right of Figure 3.3 where the variation in total rendering time is a product of variations in painting time.

Best Practice 3: Complex visualizations that encode paths can have sub-linear relationships with size and visual marks—small size changes in a bounding box may not drastically affect the rendering time.

3.3.3 Visual Marks

In this part, we discuss the effects of different styling mechanisms with respect to visualization performance. After some initial testing, we collected a set of stylistic combinations that might be explored in SVG visualizations, where we visualize scatterplot and parallel coordinate plot with different values for shape-rendering (Fig. 3.4), opacity (Fig. 3.5), radius for scatterplot (Fig. 3.6), stroke for scatterplot (Fig. 3.6), stroke-width and stroke-dasharray for parallel coordinate plot (Fig. 3.7), as well as a combination of multiple styles for both visualization types (Fig. 3.6)

3.3.3.1 Styling: Shape Rendering

Fig. 3.4 shows difference in rendering time when applying (*optimizeSpeed*, *crispEdges*, and *geometricPrecision*) compared to the standard (Table 3.1) for the respective visualizations. For scatterplots (Fig. 3.4), we see an increase in rendering time when applying these styles: *optimizeSpeed* and *crispEdges* result in 13.9% and 15.1% increase in rendering time respectively, mainly due to longer painting time. *GeometricPrecision* introduces a smaller 5.5% increase in rendering time, as a result of increase in both DOM manipulation time and styling time. It's worth noting that adding *GeometricPrecision* styling is similar to the standard, where the shape rendering attribute is unset. The reason is because shape rendering defaults to an auto setting, which means the browser decides the tradeoff between the three options but prioritizes *geometricPrecision*. For parallel coordinate plots (Fig. 3.4 (b)), there are only very small changes in rendering time—*optimizeSpeed* and *crispEdges* decrease rendering time by 3.4% and 2.8% respectively but *geometricPrecision* increases rendering time by 3.5%.

Best Practice 4: Different types of visual representations have different performance changes w.r.t. shape rendering. You may have to consider chart types when dealing with this. Specifically, circles can suffer from setting shape rendering to optimizeSpeed, which counter-intuitively results in significant increase in rendering time.

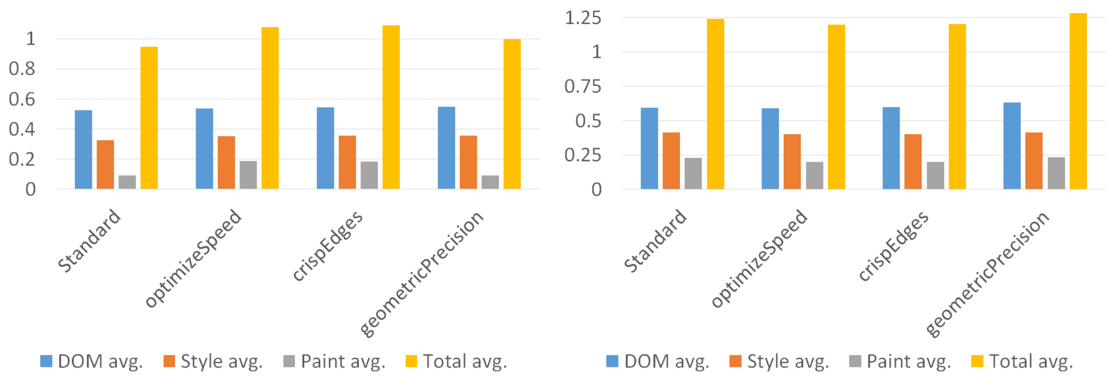


Figure 3.4: Scatterplot and parallel coordinates: Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering for the standard and when setting shape-rendering property.

3.3.3.2 Styling: Opacity

Fig. 3.5 shows the difference in rendering time when applying different opacity values for different visualizations. For both figures, the standard does not explicitly set the opacity (default to 1), and this setting also has the fastest rendering time. Interestingly, parallel coordinate plots again shows small increase in rendering time between 0.6% and 3.0%. In contrast, scatterplot visualizations show steep increase in rendering time—all as a result of increased painting time—when applying any opacity other than 1 (full opacity). Applying an opacity of 0 results in an increase in rendering time of 165.1% and all other values higher than 0 and less than 1 results in an increase in rendering of between 265.5% and 266.6%. This is a very large increase, especially because conducting the same measurements with "hardware acceleration disabled" results in up to around 30% increases in rendering time (and overall faster rendering time), which indicates a sub-optimal implementation when using the GPU to visualize SVG circle elements with non-default opacity values.

Best Practice 5: Setting a non-default opacity value for circles in scatterplots results in steep increases in rendering time, whereas for paths in parallel coordinate plots there is not a large difference. The default opacity may have the best performance and it may be better than explicitly set the property.

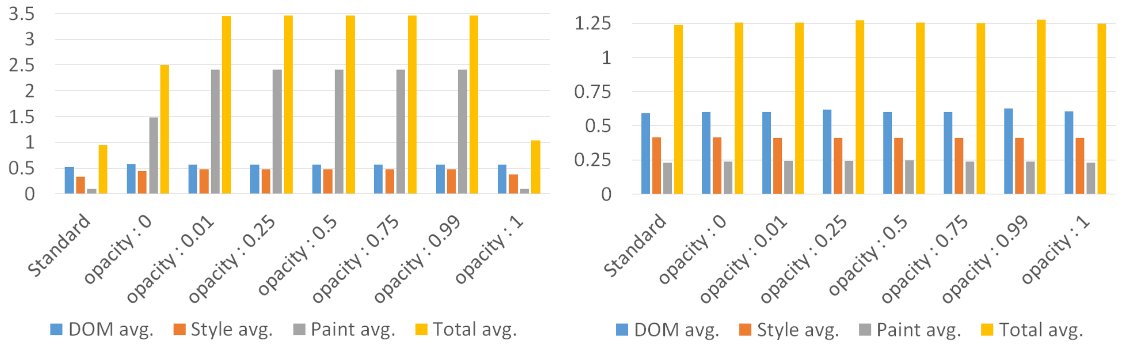
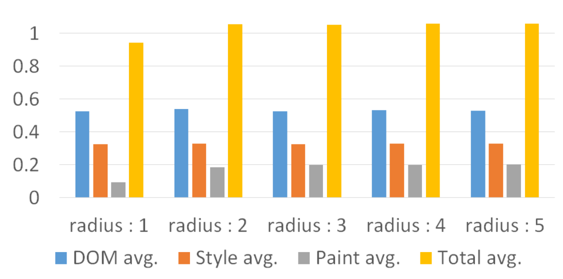


Figure 3.5: Scatterplots and parallel coordinates: Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering for the standard and when setting opacity to the listed values.

3.3.3.3 Scatterplot: Radius and Other Styles

We also consider varying radius of circles in scatterplots (Figure 3.6) in this section because perceptually changing the radius is related to styling attributes.



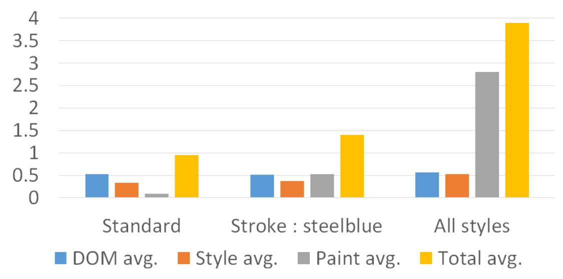


Figure 3.6: Scatterplots: Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering for Left: the standard and when setting the radius (size) of circles, Right: a) the standard, b) when adding a stroke to each circle, and c) using a combination of stroke, opacity to 0.5, radius to 5 and shape-rendering to optimizeSpeed.

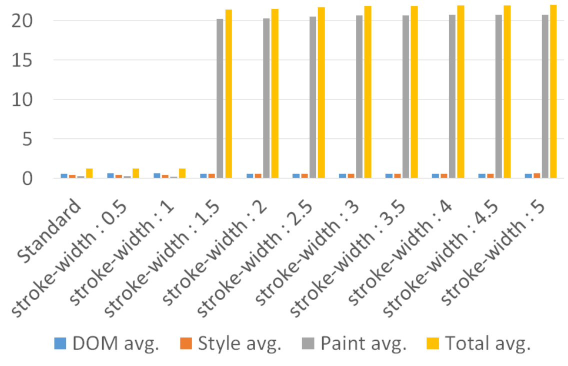
Unsurprisingly, increasing the radius of the circles increases rendering time as a result of an increase in painting time. From a radius of 2 pixels to 5 pixels there is a small, but rather consistent, increase in rendering time between 11.4% and 12.2% compared to the standard of a radius of 1 pixel.

The last figure for scatterplot styling (Fig. 3.6 right) compares the standard setting (left group of bars) with adding a stroke (set to steelblue) to circles in the scatterplot (middle group of bars) and adding a combination of stroke (steelblue), opacity (0.5), radius (5), and shape-rendering (*optimizeSpeed*) (right group of bars). Adding a stroke results in 47.6% increase in rendering time which is mainly due to an increase in painting time. Adding the above set of styles, results in an increase in rendering time of 312%, which is largely a result of painting time increase. The total increase (312%) of adding these styles combined is faster than the sum of adding the styles individually (440%).

Fig. 3.7 shows the trend in the rendering performance of parallel coordinate plots when varying the stroke-width styling property linearly in increments of 0.5

pixels. The standard that does not have stroke-width set (defaults to 1 pixel) has a near identical rendering performance with setting the stroke-width to either 0.5 and 1 pixel. Increasing stroke-width beyond 1 pixel, however, results in a massive increase in rendering time due to an increase in painting time ranging between 1624.6% and 1673.2% for all stroke-width values of 1.5 pixels and above. Interestingly, as with the opacity values for circles in scatterplots, painting time when rendering stroke-widths above 1 pixel for paths in parallel coordinate plots is much faster with "hardware acceleration disabled"—more than 3 times faster in this case. Likely, rendering time of scatterplot visualizations will increase if circles are made so large that they commonly overlap multiple of the browser's painting rectangles. However, as this is an unlikely setting in a normal scatterplot, we have not tested this.

3.3.3.4 Parallel Coordinate Plots: Stroke Width and Other Styles



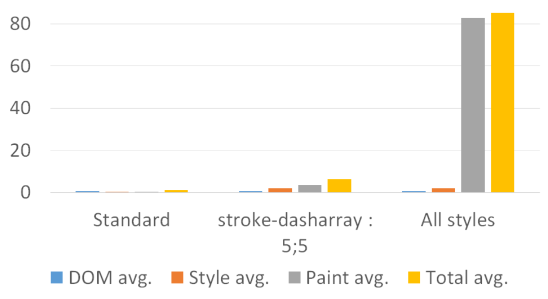


Figure 3.7: Parallel coordinate plots: Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering: Left: the standard and when setting the stroke-width of the paths/lines; Right: 1) the standard, 2) when adding a stroke-dasharray value to 5;5 to each path, and 3) using a combination of stroke-width set to 5, opacity set to 0.5, stroke-dasharray set to 5;5, and shape-rendering to optimizeSpeed.

The last figure for parallel coordinate plots styling (Fig. 3.7 Right) compares the standard setting (left group of bars), with adding a stroke-dasharray (paths) in the parallel coordinate plots (middle group of bars), and adding a combination of stroke-width (5), opacity (0.5), stroke-dasharray (5;5), and shape-rendering (*optimizeSpeed*). Adding a stroke-dasharray (5;5) results in 394.4% increase in rendering time, which is mainly a result of styling time and a painting time increase. Adding the above-mentioned combined set of styles results in a massive increase in rendering time of 6799.5%, which, despite a similar increase in styling time, is mainly a result of an increase in painting time. Interestingly, this shows a different pattern compared to the combined set of styles for scatterplot, where the combined style set for parallel coordinate plots results in significantly longer rendering time 6799.5% of the combined set than each style applied individually, which results in a total increase of 2166.5%.

Best Practice 6: Circle radius of marks in scatterplots should be a single pixel for optimal performance. However, if larger circles are needed, there are no documented performance gains from varying radius of 2 pixels and above.

3.3.4 Precision: Number Rounding

Fig. 3.8 shows the total time for rendering with and without number rounding for both scatterplots and parallel coordinate plots, respectively. The setup rounds all pixel coordinates to integers using JavaScript's standard `Math.round()` function. Rounding decreases rendering time by 10.1% for scatterplots and 12% for parallel

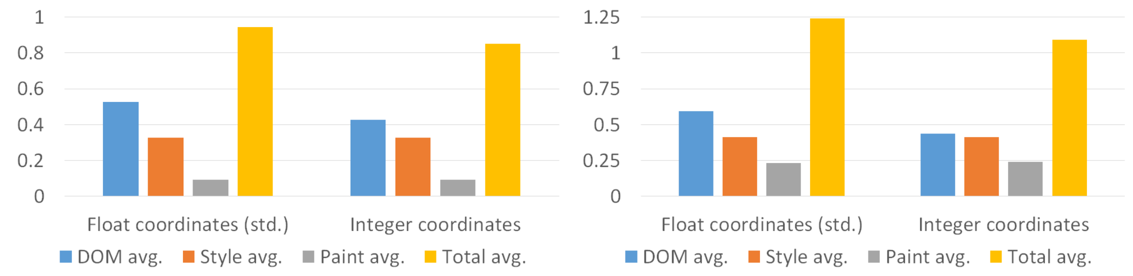


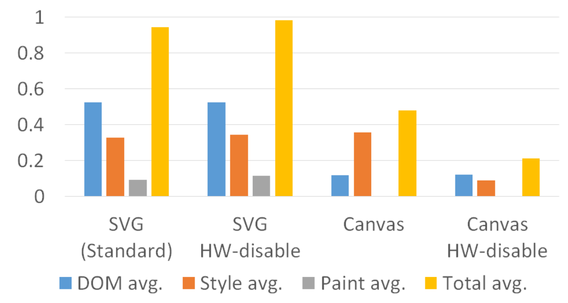
Figure 3.8: Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering scatterplots and parallel coordinates with and without rounding coordinate precision to integer values.

coordinate plots, which for both is largely a result of faster DOM manipulation time. The reason is that floating numbers with many digits increases the complexity of DOM manipulation, such as fixing the positions of the data point and axis, while using integers simplifies the process. Although rounding means loosing decimal precision, the lost precision is not necessarily notable because visual marks map to pixels when rendered on a screen. And the benefit of giving up decimal precision, yields a considerable performance gain.

Best Practice 7: Raw floating numbers increase DOM manipulation time while rounding to integers improves performance significantly. When rendering visualizations that do not need high precision, use integers to improve rendering performance.

3.3.5 Hardware Acceleration and Canvas

Lastly we take a look at rendering performance when using SVG and HTML5 Canvas elements with and without hardware acceleration enabled. Fig. 3.9 shows the performance measurements for scatterplots and parallel coordinate plots respecti-



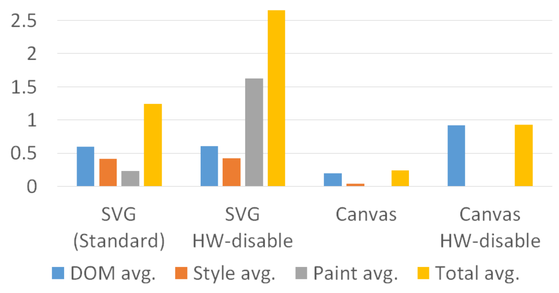


Figure 3.9: Time taken in seconds (y-axis) for DOM manipulation, style computations, and painting, as well as total time for rendering scatterplots and parallel coordinates with SVG and HTML5 Canvas elements both with hardware acceleration disabled and enabled.

vely. When creating the canvas visualizations, we have created visualization that visually similar to the standard SVG based visualizations. However, elements and styling are not one-to-one equivalents between canvas and SVG. Therefore, the total time is comparable, but the individual steps (DOM manipulation, styling and compositioning, and painting) are not individually comparable between the two types.

Disabling hardware acceleration for SVG scatterplots (Fig. 3.9) results only in a very minor increase in total rendering of 0.04 seconds. Disabling hardware acceleration for SVG parallel coordinate plots (Fig. 3.9), however, yields a steep increase in total rendering time. This is almost exclusively a result of an increase in painting time of 704.4%. While the effect of hardware acceleration is interesting, we do not include a best practice concerning it because it is not an option that a developer of web-based visualizations typically has control over.

When using canvas elements for rendering the visualizations, the painting time is near zero and therefore ineligible in Fig. 3.9. This is because the compositing of the canvas is performed in previous steps, and the only thing that is left for the browser to paint are bitmaps. For both visualizations we can observe a significant decrease in total rendering time when using canvas elements rather than SVG elements for rendering the visualizations. For scatterplots the total rendering time decreases 49.4% with hardware acceleration enabled and 77.6% with hardware acceleration disabled. The latter is a curious result that we cannot explain. However, we can add the note that during our test setup exploration phase, we observed similar instances where disabling hardware acceleration is faster when testing different styling properties. However, since, as noted above, disabling or enabling hardware is not a setting the developer has control over, we have not investigated this further. For parallel coordinate plots the total rendering time decreases 80.8% with hardware acceleration enabled and 25.1% with hardware acceleration disabled. These measurements are more in line with the intuitive expectation that having hardware acceleration enabled is faster than having it disabled.

Best Practice 8: Using an HTML5 Canvas element for visualizing data results in a significant performance improvement for parallel coordinate plots, but only a minor rendering performance improvement for scatterplots. Therefore, if declarative SVG graphics are not needed, then they can be replaced by imperative bitmap graphics with equal or better rendering performance.

3.4 Techniques: Large-Scale SVG Visualization

In this section, we describe techniques for managing large datasets and discuss the performance and responsiveness benefits on scatterplots and parallel coordinate plots. These techniques are related to sampling, data aggregation, and progressive rendering, which differ fundamentally in how they are applied to datasets—sampling visualizes a subset of the data, aggregation visualizes all data but with less detail, and progressive rendering visualizes all data slowly but with high responsiveness. The sampling, aggregation, and progressive rendering techniques are not exclusive and can be combined and applied in conjunction to facilitate responsive interaction, such as done in the Scribble Query interaction technique [101].

3.4.1 Sampling

Data sampling to improve responsiveness is a process of selecting and visualizing a subset of a dataset. Technically, the motivation and result is straightforward—rendering fewer elements results in faster rendering time, as seen in Fig. 3.2. However, sampling requires careful deliberation because rendering a subset of a dataset means losing important details of the data.

One way to mitigate this is to investigate the distributions of data variables and select representative samples following similar trends including minima/maxima and other statistics. This approach can be relatively painless in a scatterplot, where dots signify the pairwise relationship between just two data dimensions. However, sampling of paths in parallel coordinate plots is more complex, as investigated in-depth by Dasgupta and Kosara [102], who introduced Pargnostics for layout management based on screen-space metrics. Furthermore, sampling can alleviate overplotting and clutter as discussed in detail by Heinrich and Weiskopf [103] as well as Ellis and Dix [97] for parallel coordinate plots, and by Bertini and Santucci [104] for scatterplots.

To instrumentalize sampling techniques for SVG visualizations, it is important to consider a visual budget denoting a maximum number of elements to visualize or the delay time that a target user can withstand. As discussed in previous section, this will differ across device, browser, and type of visualization. However, using our web service (details in next section), it is possible to determine a visual budget for a visualization that can be approximated in an actual implementation. We say "approximate" as it is impossible to determine criteria for sampling (e.g., retaining minimums/maximums and distributions) that will guarantee a sample size, especially for high-dimensional datasets.

3.4.2 Aggregation

Like sampling, aggregation improves performance by rendering fewer elements. However, aggregation seeks to visualize all data in a dataset, albeit with a loss of detail as similar entries are grouped or discretized. Frequency in a group can then be indicated by size, color, and opacity of a visual mark. Discretization, or binning, of a dataset can be achieved while retaining some other statistics, such as averages and extrema, as investigated for parallel coordinate plots by Novotny and Hausner [105].

As two-dimensional data for a scatterplot is binned, the uniform dots are transformed into simple circular glyphs whose size denotes the number of entries in each bin. Because binning for a scatterplot is performed on two dimensions, it is possible to perform a relatively fine-grained aggregation while retaining high performance.

Aggregation of data for parallel coordinate plots needs to be much more aggressive than for scatterplots. This is simply because many more data dimensions in parallel coordinate plots usually lead to significantly higher combinations of bins. This means that the time taken to render an aggregated multidimensional dataset quickly approaches—similar to the aggregated scatterplot—a time maximum similar to the rendering time of non-aggregated data. Rendering time will, however, vary greatly with the number of dimensions in the data.

Charts	**Elements**	**Time in seconds**	**Increase over std.**	**Pri. contr.**
Scatterplot	5,000	28.2s	2880.4%	Style or paint time
Par. coor.	5,000	78.8s	6260.6%	Style or paint time

Table 3.2: Total rendering time of scatterplots and parallel coordinate plots rendered progressively, the percentage increase over the standard and the primary contributors to the increase. Note that the percentage increase over the standard compares progressively rendering 5,000 elements with batch rendering of 50,000 elements in the standard.

3.4.3 Progressive Rendering

Progressive rendering ensures responsiveness in rendering data without loss of detail by rendering parts of a visualization at a time. This means that the browser completes a full circle of DOM manipulation, styling, and painting for each visual mark. Table 3.2 shows measurements for progressively rendering scatterplots and parallel coordinate plots, both with 5,000 elements. Despite the seemingly poor performance of progressively rendering SVG visualization depicted in Table 3.2, progressive rendering is still a viable strategy because a) it prioritizes interrupting the rendering of a visualization that could otherwise take seconds and make the browser seem unresponsive to the user and b) the measurements in Table 3.2 depicts a worst case performance where visual marks are rendered one at time. To achieve faster performance one could, e.g., render 20 or 100 elements at a time to achieve another tradeoff between responsiveness and total rendering time. In practice, progressive rendering is implemented using a function for manually iterating through an array of data elements to visualize. However, instead of self-instantiating the function, the function sets a timeout with 0 ms delay, as depicted simplified in Listing 3.1. The timeout facilitates that the visualization iteration can be terminated at each iteration, e.g., due to user interaction, thus ensuring responsiveness. The idea to progressively render visualization to show partial visualizations while allowing for user interruption is related to ProgressiVis [90]. However, the progressive rendering technique presented here (in Listing 3.1) differs from ProgressiVis because our technique is implemented exclusively on the client.

```
var svgParent = d3.select(body).append("svg");
var timeoutID;

function appendElement(data, index) {
  if (index === data.length) {return;}
  svgParent.append("elementName")
    .datum(data[index])
    ...;
  timeoutID = setTimeout(function () {
    appendElement(data, index + 1);
  }, 0);
}
appendElement(data, 0);

function cancelTimeout() {
  clearTimeout(timeoutID);
}
```

Listing 3.1: Simplified JavaScript implementation of progressive rendering.

3.5 Discussion

Every technology has some bottlenecks in its implementation. Therefore, our findings reveal the practices that work best with the current state of web browsers. However, as versions progress in the future, we expect some changes in the results of our evaluations.

First of all, since we investigate client-side renderings of datasets, rendering is limited to less than 100,000 data points, which is beyond the SVG rendering capabilities of most current browsers. In contrast, some web visualization systems have tackled billions of data items [22] using databases and server-side technologies. However, even in such systems, the browser is left to deal with orders of a few thousands of data items to render.

When we started our investigation of rendering performance of visualizations, we expected to just see that rendering time changes linearly with number of data items used for the visualizations seen in this section. However, we were surprised to discover that there is a much more nuanced relationship between browser internals and the type of visual marks and their styling being visualized. This underlines the need for techniques that can visualize large datasets with a sub-linear relationship between the number of elements and the time taken to render them, because naïvely visualizing large datasets in browsers using SVG can cause serious performance implications.

Finally, as with most optimizations, many of the techniques outlined discussed here are somewhat ad hoc, low-level, and even “dirty” in that they take advantage of intricate knowledge of web browser rendering that goes beyond the standard APIs exposed to the programmer. For example, forcing a programmer to use only integer coordinates to achieve acceptable rendering performance for a large-scale visualization is crude and fraught with many disadvantages. It could be argued that the onus is on browser developers to optimize their rendering implementations so that client programmers can use the APIs without having to rely on such specialized knowledge. These are valid points, but at the same time, visualization developers are often charged with making a specific visualization work with present technology and can rarely afford to wait for said technology to improve. For this reason, we believe that the techniques described in this chapter, while not theoretically elegant, may still have significant real world impact.

3.6 Conclusion

In this chapter, we have presented a comprehensive evaluation of rendering performance for SVG visualization. Based on in-depth investigations of browser rendering performance of SVG visualizations, we have described the nuanced relationship between browser internals, dataset size, and type of visualization in detail. We provide and evaluate a set of sampling, aggregation, and progressive rendering techniques to operationalize our findings. While some other approaches are capable of visualizing much larger datasets [22] than the datasets used in our evaluations, there currently exists little work that details the nuanced relationship between dataset size and rendering time when creating SVG visualization in a browser.

Chapter 4: Supporting Web-based Visualization through Ad-hoc Computational Clusters of Multiple Devices

In this chapter [1], we present VisHive (Fig. 4.1), a JavaScript framework for creating ad-hoc, opportunistic clusters consisting of local, networked devices that are directly integrated in a web application. VisHive is developed to leverage available computational resources in the situation where a user is engaged in a web-based visualization with access to multiple devices in their immediate physical surroundings. For example, if the user is accessing the visualization using a laptop, they may also have a smartphone in their pocket, a tablet in their backpack, and a personal computer in their office. While offloading computation to a server-side or cloud-based component is certainly possible, it would make a lot of sense if the user was also able to fire up their additional client-side devices and use them for opportunistically offloading any heavy computation required by the visualization tool. Specifically, if a user is analyzing a huge dataset that takes a lot of time, he/she would probably think of using cloud or server to do this, but the server may not be immediately available and even it's reachable, it still needs a lot of time to set up connections, data import and export interfaces, etc. The user may give up

[1]This chapter is an adaption of the paper [106] in Information Visualization Journal.

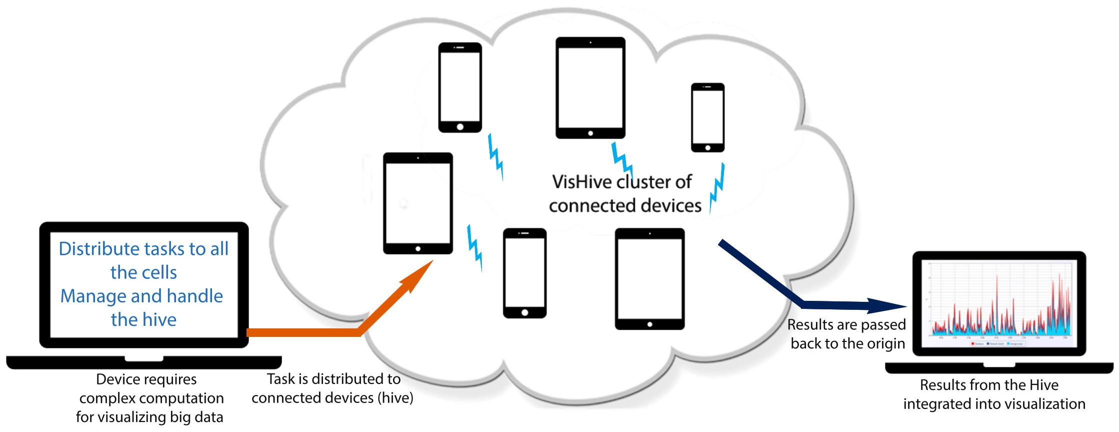

Distribute tasks to all the cells
Manage and handle the hive
Device requires complex computation for visualizing big data
Task is distributed to connected devices (hive)
VisHive cluster of connected devices
Results are passed back to the origin
Results from the Hive integrated into visualization

Figure 4.1: VisHive creates ad-hoc and opportunistic clusters from the local devices available to a user. Here, laptops, smartphones, and tablet devices are connected into a cluster to handle complex computations. Connected devices contribute computational power using VisHive.

the idea due to the complexity. On the other hand, if the user can utilize their idle devices to help the computation of visual analysis without worrying about set up, it's much easier and simpler.

Compared to the server-based or cloud-based solution, this "local cloud" of co-located physical devices brings benefits to both end-users and developers. More specifically, the end-user can avoid any mobile network fees and minimize latency by confining the communication to the Local Area Network (LAN), whereas the developer can implement concurrent computation using JavaScript in the visualization client and without having to worry about deploying a separate service for this purpose.

To demonstrate the utility of VisHive, we present four examples of web-based visualizations using the toolkit instantiated from the framework to implement computationally expensive distributed algorithms: Wikipedia text analytics visualization, incremental database query [43], distribute DBSCAN clustering, and a distributed PCA algorithm. The evaluation shows that there is a significant time improvement using the VisHive framework on various combinations of devices—laptop, tablet, and smartphone, compared to single-device computation.

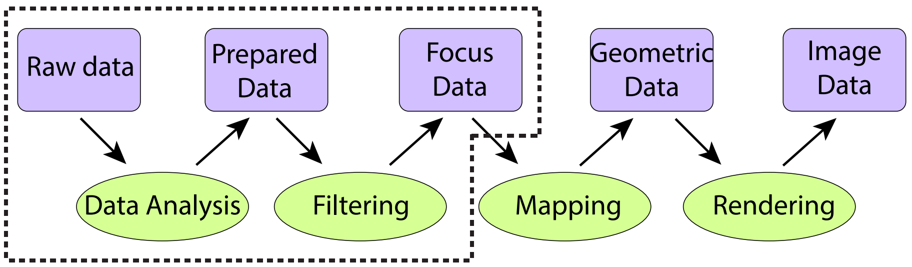


Figure 4.2: Basic visualization pipeline [107] (from `http://www.infovis-wiki.net/index.php/Visualization_Pipeline`). Data transformations, rendering, and view transformations can be processed in a distributed manner. The stages within dashed bounds are those that the proposed parallelism focuses on.

4.1 Design Guidelines

The web is becoming a ubiquitous medium for sensemaking through visualizations [19], sharing visual insights from data, and harnessing collective intelligence. [108] However, there currently exists no satisfactory mechanism for executing computationally intensive algorithms commonly needed for visualization and visual analytics on the local client. As discussed in the previous section, the goal of our ad-hoc computational cluster framework is to facilitate the creation of ad-hoc device clusters using standard web technologies. The driving scenario behind the framework is the fact that people today tend to carry more than a single device with them at all times. Leveraging these devices together can help scale our analytics applications to the challenges of big data.

In general, visualizations follow a transformative pipeline that turns data into interactive graphical representations through multiple stages. [98] To target visual analytics of big data, we need distributed frameworks integrated with the visualization pipeline using connected local devices to generate a visual representation and handle user interaction. For this purpose, below we list seven design guidelines driving the ad-hoc computational cluster framework.

4.1.1 Networked Devices

The fundamental requirement for a distributed system is a network of connected nodes. Thus, the framework should be capable of connecting multiple devices into a distributed system.

D1 *Cross-platform support:* The devices used by analysts for personal computing and sensemaking can be diverse, ranging from personal computers to mobile devices. Therefore, the framework should work independent of the underlying platforms, modality, and physicality of these devices.

D2 *Ad-hoc connectivity:* A user should be capable of opportunistically creating a cluster from available devices. This includes adding to or removing devices from the clusters at any point.

Peer-to-peer networks are ideal for this purpose [109, 110] as they do not set a hierarchy among the devices, and they do not require a dedicated server infrastructure to create clusters.

4.1.2 Responsive Distribution

Once the devices are connected into a distributed system, supporting computation on the device cluster requires intelligent management of the connected devices. The challenge in this case is to ensure that adding or removing devices at any point does not interfere with user activity within the visual analytics system.

D3 *Responsive computation management:* All available devices should be free to contribute processing power to computational activities. Computation jobs assigned to devices within a cluster should not only be based on the processing power and available memory on the device, but also based on their current use.

D4 *Fault-tolerance:* Devices entering the cluster should immediately be assigned new jobs, and devices leaving it should be able to return a job unfinished so that other devices may take up the remainder of the job. This mechanism should also be robust in the face of device or network failure.

4.1.3 Supporting Visualization and Interaction

Visual analytics systems often utilize computationally complex algorithms. For example, browsing histories of users can be used to generate and visualize spanning trees in order to understand their web traversal history. [111] Machine learning and data mining models are also used to identify specific features, visualize interesting patterns, and prompt user exploration. [112, 113] While some of these models are inherently parallelizable in their logic, it should also be possible to configure how the underlying algorithm can spread across the clusters of varying sizes and resources.

The data transformation, rendering, and view transformations are the basic data manipulation processes in the visualization pipeline (Fig. 4.2). Our goal is to distribute tasks to the whole computational cluster and make the processes parallel within the pipeline, to reduce the overall delay of visualization systems.

D5 *Distributed processing:* Algorithms for distributed processing, such as MapReduce [114], should be applied to chunks of data across the ad-hoc clusters. The framework should also support defining a distributed version of an algorithm at each stage of the visualization pipeline (Fig. 4.2) with features to adapt the algorithm to the specific cluster.

D6 *Data-driven distribution:* The distribution of jobs to multiple devices should be adapted to the dataset itself based on the attributes, data types, and sources. Computations in the visualization pipeline involve transforming data of one form (input) to another (output) at each step. Similar to popular big data systems (e.g., Hadoop HDFS [115], Google BigTable [116]), it should be possible to create job chunks for devices in the cluster by splitting any dimension of the data. For example, in spatiotemporal data, jobs can be created either by splitting data based on time or space, in order to reveal incremental details in the visualization when the data is being processed by the cluster.

D7 *Handling user interaction*: User interactions are essential for interactive visualization in visual data analysis. Interaction steers the visualization pipeline to focus on specific data subsets and encodings to promote focused visual analysis. Specifically, corresponding computations should respond to user interactions; outdated computations should be stopped and new computations should be started based on the user's focus conveyed through interaction.

4.2 Challenges and Contribution

With the rise of big data and increasingly sophisticated analysis methods, *scale* remains the dominant computational challenge for visualization. Put simply, the bandwidth, memory, and computational demands of modern data problems are often too large for a single workstation to manage. These challenges are exacerbated by the fact that visualization is increasingly being moved to the web [17] and thus no longer have full access to the computational power of a desktop computer; in fact, with the proliferation of mobile computing, it is even more likely that a visualization is viewed on a mobile device such as a tablet or a smartphone than a personal computer altogether. [117]

The standard solution for resource-hungry visualization applications is to turn to client/server solutions, where a thin client in the user's browser offloads the bulk of any computation to a server with significant capacity. However, in this proposal, we propose a complementary solution based on opportunistically creating ad-hoc computational clusters utilizing local devices in the vicinity of the user. Below we discuss the strengths and weaknesses of both approaches.

4.2.1 Standard: Cloud or Server-based Computing

If the visualization client is insufficient for a resource-heavy computation, the standard solution—particularly for web-based ecologies, where there already is a server infrastructure in place—is to offload the computation to a server on the Internet (or in the cloud). This requires the use of server-side middleware, such as Node.js[2], Flask[3], or Ruby on Rails[4], which will communicate with the client using protocols built on top of HTTP.

- **Strengths:** Flexible, powerful, and standardized.
- **Weaknesses:** Non-trivial setup, prior planning, potentially costly, security concerns.

4.2.2 Novel: Ad-Hoc Computational Clusters

Our main contribution in this work is *ad-hoc computational clusters* on the client that take advantage of opportunistic ecosystems of devices in the near vicinity. The goal is to simply leverage the idle computing power of these devices to mitigate scale for visualization computations. By virtue of integrating this distributed computing capability within the visualization client itself, our framework provides a tighter loop that allows for several parts of the visualization pipeline to be offloaded onto multiple devices.

[2]`http://nodejs.org/`

[3]`http://flask.pocoo.org/`

[4]`http://rubyonrails.org/`

- **Strengths:** Lightweight, no setup, no downloads, no prior configuration, leverages existing and idle computing power.
- **Weaknesses:** Limited in scale, bandwidth-intensive, requires distributed computing knowledge on behalf of the visualization programmer.

4.3 Framework Overview

The VisHive toolkit was developed for building ad-hoc and opportunistic clusters of computing devices for web-based visualization. It is implemented completely in JavaScript to target the web platform, thus providing cross-platform support (D1). It uses the WebRTC standard by W3C[5] for establishing peer-to-peer connections across web browsers. Since the web is the target platform, the devices—called *cells*—are connected into a device cluster—known as a *hive*—as soon as they open a VisHive application webpage on the web browser (D2). The toolkit provides modules for structural definitions of distributed algorithms based upon the attributes of the hive (D3), and handles entering/leaving cells in the hive (D4). The toolkit integrates closely with the visualization pipeline, allowing developers to handle the stages in the pipeline in parallel using the connected devices (D5, D6, D7). Figure 4.3 shows the network architecture of an VisHive toolkit example. The VisHive toolkit is open-source and can be accessed online [6].

[5]`http://www.w3.org/TR/webrtc/`

[6]Website anonymized for double-blind reviews.

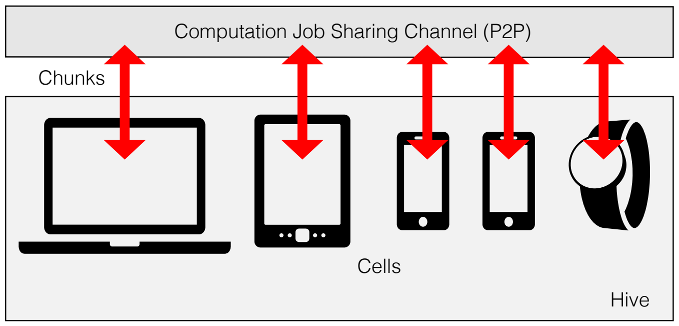


Figure 4.3: Example VisHive application network architecture.

4.3.1 System/Network Architecture

The VisHive toolkit consists of five components to fulfill the design requirements above (Figure 4.4):

C1 **Job partition layer** that divides a high-level computation operation into computation jobs (*chunks*);

C2 **Communication layer** to share chunks across cells;

C3 **Integration layer** that combines the results from all cells and passes them to the web visualization;

C4 **Job control layer** handling cells entering and leaving the hive (fault tolerance); and

C5 **Matchmaking service** that connects multiple devices in a specific physical space into clusters (hives).

Figure 4.4 depicts the VisHive architecture with these components. VisHive uses a peer-to-peer (P2P) network architecture established across the browsers of the cells using WebRTC technology, popularly used for real-time video calls over the web browser [7]. Our implementation uses the open source PeerJS framework [8] for establishing peer-to-peer connections across the cells. The P2P connection creates the communication layer (C2) for transferring chunks to the cells within the hive. Only the matchmaking service (C5) is centralized and requires a dedicated server component (this is commonplace for many peer-to-peer applications); other components are based solely on standard web technologies. Providing a centralized matchmaking server is easy and can be achieved with scalability to serve a large number of hives.

The control of the distributed system lies inherently with the instance that the user actively interacts with. The toolkit is not designed for collaborative visualization; thus, VisHive supports just one active user interacting with a distributed application on a device. This way, the controlling instance, or *master*, takes the help of other idle devices, or *slaves*, to share computations amongst them. One thing to note here is the difference between a typical P2P architecture and our implementation. While the devices are connected by the P2P network, the VisHive master keeps track of computations assigned to each of the slaves to collect the computed results back from them. The master therefore manages the splitting and sharing of computations. This structure is resilient as it takes advantage of the P2P connection, while flexibly allowing any device to act as the master based on the user's focus. In general, if the user focuses on a device, results are expected to be shown

[7]`https://apprtc.appspot.com/`

[8]`http://peerjs.com/`

on that device, so it acts as the master.

4.3.2 Matchmaking and Communication

Hives are initialized on the matchmaking service, a modified web server built in Node.js that typically runs on a local device such as a laptop or, alternatively, on a remote cloud-based server. The first device to connect to the hive automatically becomes the master; this can be manually changed. Additional cells are connected by navigating their browsers to the matchmaking URL, thus adding them to the peer-to-peer communication channel. As these cells join the hive, they share details on their capabilities based on the client and operating system.

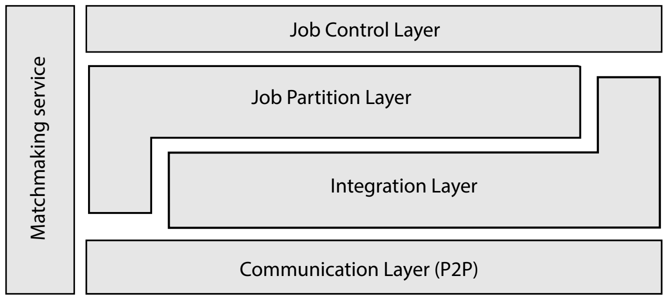


Figure 4.4: VisHive toolkit infrastructure containing five components to create and manage distributed computation jobs (chunks).

The matchmaking server only manages the peer-to-peer session for the hive. It does not handle data management, job allocation, or computation. These are the responsibility of the master, which is a special cell. Since the VisHive toolkit targets ad-hoc and opportunistic device clusters (for example, between an analyst's smartwatch, smartphone, and laptop), this registration process ensures that distribution happens in an environment-aware fashion.

After the cell registration process, individual cells are capable of accepting the computation chunks involved in each stage of the visualization pipeline (C1). When a master shares computation jobs with the slaves in the hive, the cells accept the jobs and look up the input data from the job definition. Cells will then perform the required computations on the input using the shared computational models and send the output back to the master to be recombined.

4.3.3 Masters and Slaves

Regardless of whether a cell is a master or a slave, they use the same JavaScript codebase, thus making application development simple (Listing 4.1). The client programmer simply has to provide a master implementation, consisting of the visualization and interaction part of the web application, as well as a separate slave implementation, which handles the computation. The programmer also has to provide an implementation for recombining results. This follows practice in distributed algorithm design, such as MapReduce [114].

```
vishive.init(url);
peerid = vishive.getChannel();
var hive = vishive.connect(peerid,
    function (hive) {
        # Master implementation
        # ... visualization and interface setup
        # ... job distribution
        # ... manage results
    },
    function (hive, data) {
        # Slave implementation
        # ... computation on subset
    });
```

Listing 4.1: JavaScript code for initializing VisHive in a standard web-based visualization.

Due to VisHive's clear separation of concerns between masters (interface and visualization) and slaves (computation), a hive consisting of only a master would not make any progress on the computational task. In practice, VisHive allows the master to also run a slave instance in a parallel thread (web worker) to allow the application to perform the computation on the same device. This ensures that progress can be made even if no computational resources are available other than the device on which the master is running.

Note that the master-slave architecture is independent of the matchmaking service mentioned above. The matchmaking server can reside on any device within the same network. It may or may not be one of the devices in the hive. It only manages establishing the connections between participating devices. This is quite common in P2P architectures. [109]

4.3.4 Job Allocation and Control

Job allocation and control within the VisHive distributed system is handled by the C1 and C4 components of the toolkit. Each computation job (chunk) is treated as a mapping from input to output generated by shared computation logic, similar to the MapReduce model [114] for processing big data on parallel and distributed systems. The default configuration for job partitioning involves splitting the input data for a high-level computation into jobs that each slave works on parts of the data. The job allocation module creates the chunks based on the available resources on each cell and the number of cells in the hive (including the master and the slave cells).

Take mean calculation. for example. Assume the dataset has $1,000$ entries and one column data point for simplicity, and there are 4 devices available for computation. The job allocation is to split the dataset into 4 chunks (250 entries one chunk for an even split), assign each chunk to each device (sending data). Devices compute mean of the partial dataset and send results back. The actual allocation process of how to splitting data is provided as API (discussed in later section), so the user can define their own data chunks.

Explicit application logic created by the VisHive application developer (end-user developer) for splitting a computation (and input) into chunks is also supported.

4.3.5 Fault tolerance

The job control component (C4) is responsible for automatically detecting when existing cells leave or new cells enter the hive. Leaving the hive also includes device or network failure, when devices leave unexpectedly. This is detected by the master, in which case the assigned chunk is retracted and added to the top of the queue for reassignment. Similarly, a cell that enters a hive gets added to the queue of available computation cells immediately.

When a slave cell receives a job chunk to process but does not respond back to the master in a timely manner (this may be due to disconnection, node failure, or slow computation), the cell will be regarded as failing and this chunk of job will be reassigned to other available cells. In this way, VisHive deals with cells entering and leaving the hive at any time. Generally speaking, to make the system simple and easy to maintain, the matchmaker service treats all non-responsive cells as failing. This may cause duplicate jobs when one cell has network problems, causing the master to assign the job to another cell, only to have the original cell return with the result. While this does waste computational resources (for one chunk), it is an efficient way for VisHive to operate reliably.

VisHive provides all of the mechanisms for distributed algorithms, but does not actually implement any specific algorithm. Thus it is up to the application developer to manage conflicts, shared memory, concurrency, locking, and mutual exclusion. In particular, the toolkit assumes that any conflicts occurring during the integration process that are application-specific are handled by explicit application logic developed by the client programmer.

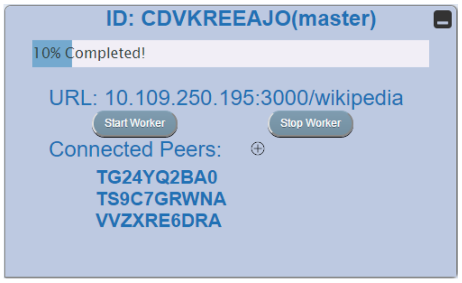


Figure 4.5: VisHive console widget showing controls and status for the hive, its cells, and the current computation.

4.3.6 Visual Interface

The *console* is the dedicated visual interface of the toolkit itself (Figure 4.5), and contains the status of all the devices within the existing computational cluster.

The VisHive toolkit is closely integrated with the visualization pipeline. Following this model, each stage of the pipeline involves transforming an input into an output. Implemented as a separate widget that can be hidden as needed, the console gives both controls as well as shows the status of the current hive, connected cells, and any ongoing computation progress. This supports monitoring progress in each of the visualization stages. For example, data cleansing involves converting the raw data into a structured data structure, which requires going through individual data points, parsing them, and processing through each cell. This can be managed through the interface. In case of large datasets, this operation can be expensive due to the sheer amount of data. VisHive can split the data into computational jobs that can be processed across the cells in the connected hive (component C1), while at the same time enabling real-time updates and control of the process.

4.3.7 Implementation Notes

VisHive is a pure JavaScript toolkit implemented using the PeerJS toolkit for peer-to-peer communication and using the D3 [19] toolkit for rendering visualizations. More specifically, VisHive events can be explicitly bound to D3 joins so that the visualization can be automatically updated when the data is loaded, a chunk is calculated, or the computation is finished. For example, the "plot" function in VisHive API (Listing. 4.2) handles D3 states (enter, update, and exit) execution. When the function is called, it recomputes the join and maintains the correspondence between elements and data [118]. In this way, visualizations in VisHive are integrated with D3 joins. With the exception of the matchmaking service, all components run directly in a modern web browser without requiring specific software. The matchmaking service can either be run locally, in which case a Node.js installation is required, or on a remote cloud server.

```javascript
# Connect to matchmaking service
visHive.connect(config, sessionId);

# VisHive event handler definition
visHive.eventHandler = {
    dataPreProcess: function(rawData) {
        # pre-process the data
        return formattedData;
    },

    splitData: function(chunkId, formatData) {
        # split the data
        return chunks;
    }

    mergeData: function (chId, chunk, mergedData) {
        # merge the results into the main result
        return mergedData;
    }

    process: function (receivedData, dataDice) {
        # compute the results on the slave
        return results;
    }

    redraw: function (data) {
        # use D3, etc for data plots.
    }
}
```

Listing 4.2: VisHive API declaration in JavaScript.

4.3.8 VisHive API

To demonstrate how to use the VisHive toolkit to aid with distributed computing for visualization, we here discuss the functions in the API that developers can override to integrate into the VisHive toolkit. Code for function declarations are in Listing 4.2. The API contains five main functions: data preprocessing, split, integration/reduce the results, job process on devices, and visualization.

4.4 Examples

To showcase the utility and the flexibility of the VisHive framework, we implemented four examples that demonstrate different common computational needs for visualization applications: (1) a distributed text analytics visualization, (2) a distributed incremental database query for exploratory visualization, (3) a clustering algorithm, and (4) eigenvector calculation for Principal Component Analysis. To detail the implementation use cases, we provide the pseudocode and explanation of the progressive text analytics visualization example.

4.4.1 Distributed Text Analytics for Large Document Corpora

Visualizing results from text analytics can reveal characteristics of and relations between articles in a document corpus. However, many information retrieval algorithms involving words frequency counting are limited due to significant processing time for large-scale document collections. This process can be made faster through multiple devices, each working on a different part of the document corpus.

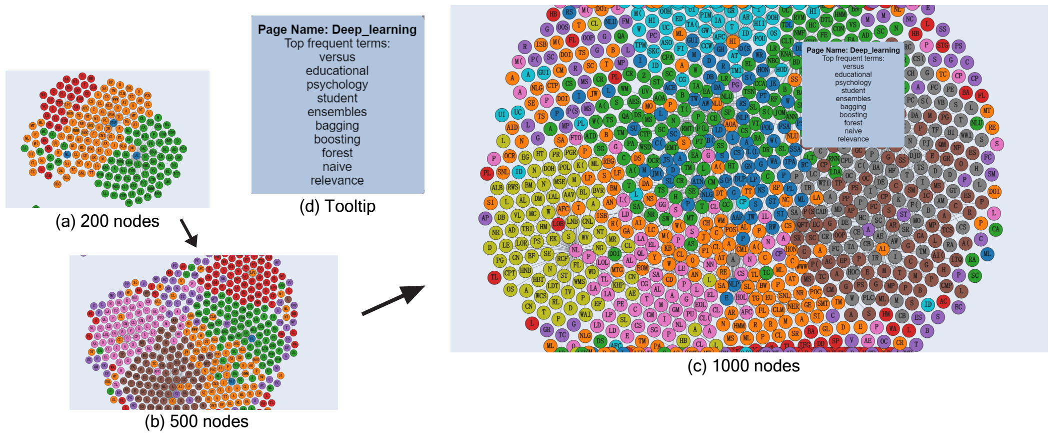


Figure 4.6: Node-link diagram visualization for different number of Wikipedia articles. (a), (b) and (c) show 200, 500, 1000 pages, respectively; (d) is the tooltip with top frequent terms for one article (deep learning); nodes are Wikipedia articles, labeled initials of article name (e.g. DL = "Deep Learning"); links represent hyperlinks between pages; mouse hover shows info on each page.

Our text visualization example is designed for visualizing Wikipedia by counting word frequencies for Wikipedia articles in a distributed manner, crawling text documents from Wikipedia web, calculating TF-IDF scores across multiple devices, and visualizing articles and their relationships using a node-link diagram (Figure 4.6). We use TF-IDF [119] for simplicity; other, more sophisticated, text analytics metrics are also possible.

Implementation: In our distributed implementation, the master assigns article links (English) from a central FIFO queue to cells in a breadth-first article crawler. Cells retrieve the articles using the Wikipedia API [9], calculates the word frequency table for the article, and identifies all of the internal Wikipedia article hyperlinks. The frequency table is returned to the master, updating the central word

[9]`https://www.mediawiki.org/wiki/API:Main_page`

frequency table as well as the TF-IDF rankings for the existing nodes. Furthermore, new hyperlinks that have not yet been crawled are added to the central queue. The corresponding node-link visualization on the master is updated with top keywords once all the results are returned from all the slaves. Master deletes the existing SVG and renders new one using D3 when new computations are finished due to user interactions.

```
# handle initial connection for each peer.
peer.on('open', function(id, clientIds) {
  # do nothing for master
  conn.on('data', function(data) { # slave
    # receive indicator from master.
    if (data == "master")
      conn.send("ready");
    else
      var tf = processFunc(data.pages, data.links);
    return data.links, tf;
  });
});
# handle peers that are already connected.
peer.on('connection', connect);

function connect(conn) {
  conn.on('data', function(data) {
      if (master)
      # receive ready from slave
        if (data == 'ready')
        # send data
        else
        # receive results, merge it
          mergeData(data, TFStorage);
      if (all results received)
        plot(data);
    }
  }
```

Listing 4.3: Pseudocode implementation for wikipedia text analytics

Figure 4.6 shows screenshots of the master visualization with 200, 500, and

1,000 nodes crawled on a laptop, where the queue has been seeded with a specific Wikipedia article. We use force-directed layout framework in D3 [19] to visualize relations between articles. Nodes represent pages and links are hyperlinks between pages. Each node is labeled with initials of the page name. Tooltips with article name and top ten keywords (TF-IDF) will show up when the mouse is hovered on the node. Listing 4.3 shows the pseudocode to handle master/slave data transferring and processing. String "ready" is sent from slave to indicate master that the device is ready for computation. Once master receives the message, it sends one chunk to the slave for processing.

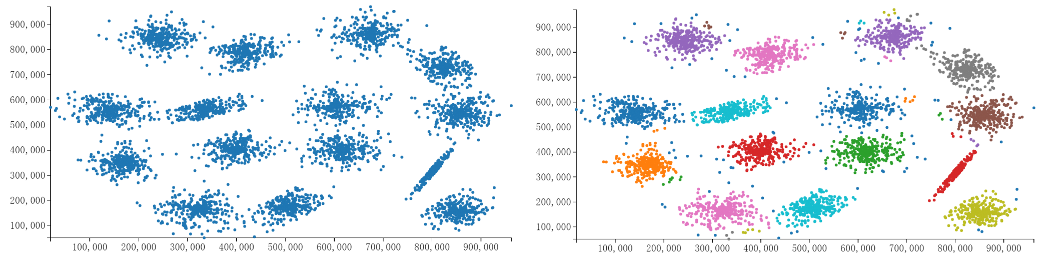


Figure 4.7: DBSCAN implementation of 5,000 points using VisHive, including before (left) and after (right) the algorithm has been applied. Different Colors represent different clusters.

4.4.2 Exploratory Visualization: Incremental Database Query

In the new era of big data, even when all of the data is available in a massive-scale database, querying the data can be forbiddingly expensive. However, many times the analyst is not interested in detailed results from a query but only need some rough idea of the contents of the data to serve as a stepping stone in the analysis [33]. For example, given a very large dataset of numeric data, the user may want to quickly calculate some descriptive statistics while discarding the actual data itself. From a visualization perspective, partial visual analysis is a quick and efficient way to catch the overview of the data. It follows Shneiderman's [120] visualization rule of "overview first, then zoom in", and insert user interaction before zooming in, which saves both computation resources and shortens the time for analysis.

Implementation: In this example, we use VisHive to implement an incremental database query based on the idea proposed by Fisher [43]. The master splits the entire dataset into manageable chunks (row indices for, say, 1,000 rows each) that can be assigned to cells that are part of the hive. A job in the cell simply consists of retrieving the chunk data, calculating some partial descriptive statistics (min, max, mean, and variance), and then discarding the data before sending back the results to the master. The master combines the results. Since typical database systems currently lack the ability to query rows by index, we avoid this restriction by using a large flat file as the database.

4.4.3 Distributed DBSCAN Algorithm

DBSCAN [121] is a density-based clustering algorithm that groups points based on their proximity. It is also one of the most common clustering algorithms since, unlike k-means, it does not require the user to specify the number of clusters a priori, it allows for arbitrarily shaped clusters, and it is robust to noise and outliers.This algorithm is one typical distributed computing in data processing stage of the visualization pipeline.

Implementation: Our DBSCAN implementation for VisHive (Figure 4.7) uses a distributed algorithm based on first computing the distance metrics (Euclidean, Manhattan, or other distance metrics) of each pair of candidate points in a distributed manner. Specifically, the master divides total points into chunks, and assigns a chunk (group of points) to a cell. Each cell computes the distance metric between the chunk of points and all the other points. The master then combines all the distance metric, computes matrix decomposition and sums up the clusters of points in the final merging stage.

Figure 4.7 shows as scatterplot of the points before and after the DBSCAN algorithm. Clusters are represented in different colors.

4.4.4 Distributed Principal Component Analysis

Principal Component Analysis (PCA) is a common approach to dimension reduction in data science that is based on projecting a high-dimensional dataset into lower-dimensional subspace using a set of values of linearly uncorrelated variables called principal components. These components are selected so that they each have a maximal variance in order to best model the data in the dataset. Determining the orthogonal components actually involves deriving the eigenvectors of the covariance matrix. redSimilar to the previous example, PCA is an important tool for data analysis and visualization since it reduces high-dimensional data to 2D or 3D data that are easy to visualize.

Implementation: Our VisHive implementation of distributed PCA splits the entire matrix on the master based on rows and participating cells compute the partial covariance matrix for sub-matrices. This can be achieved using SVD or eigenvalue decomposition. The master will finish the algorithm by estimating the whole covariance matrix based on results of sub-matrices, computing the global principal components, and choosing the first k dimensions that the cells can utilize in projecting chunks of the dataset in a second distributed phase.

4.5 Performance Evaluation

We evaluated the VisHive toolkit using our four example implementations from previous section. In order to study the impact of concurrent computation, we varied the device hardware conditions for the cluster and measured the total completion time. The WiFi used is the standard high speed university wireless network. One thing to note is that we use web worker (multi-threading) in all the evaluations with laptop so as to enable task running on the master. Table 4.1 shows the performance results in seconds.

Our four examples had the following dataset conditions:

- **Wikipedia Text Analytics:** 1000 Wikipedia web pages;
- **Incremental database query:** 200,000 rows and 10 columns of floating point values stored in a flat file;
- **DBSCAN:** 5,000 two-dimensional floating point values; and
- **Distributed PCA:** $10,000 \times 200$ floating-point matrix.

The hardware used in these experiments was the following:

- **Laptop 1 (master):** a Windows laptop with 4 Intel core i7 CPUs and 8 GB of memory;
- **Laptop 2:** a MacBook Pro with 4 Intel core i7 CPUs and 16 GB of memory; and
- **Smartphone:** a Huawei Ascend 7 Mate running Android with a HiSilicon Kirin 925 CPU (four Cortex-A15 cores).
- **Tablet:** a Samsung Galaxy S 10 with Quad-core Krait 400 CPU.

Algo.	**1 LT**	**1 LT + 1 Ph**	**1 LT + 1 TL**	**2 LT**	**2 LT + 1 Ph**
Wiki node link	220	182	168	135	98
Database query	23	18	17	14	10
DBSCAN	150	120	112	85	60
PCA	59	46	43	36	27

Table 4.1: Computation time (in seconds) for our four different example implementations for five different device combinations involving laptops, smartphone, and tablet. LT: laptop, Ph: phone, TL: tablet.

As can be seen from the performance results in Table 4.1, there are significant improvements in completion time when involving additional devices beyond the initial laptop master. In particular, when three devices are involved, the completion time is less than half of the original for all four examples. We take this as an indication that the overall idea and current implementation behind the VisHive toolkit is sound.

In addition, we instrument our code to measure the actual time spent in computation and data transfer (includes sending data and returning results, etc) for different devices. For laptops, the average ratio of transferring time over total computation time is between 10-20%, whereas it is 25-35% for smartphones. We also evaluated the performance by adding more smartphones into the hive to detect data transferring overhead. When 3 or more smartphones are involved, the overall computation time does not increase significantly due to the heavy transferring time on mobile devices. The time is mainly determined by network situations and I/O speed, and this varies across devices and networks. Since smartphones have much smaller I/O throughput, when data becomes larger, I/O constraints will hinder massive deployment of the framework. These limitations are discussed in discussion section.

4.6 Discussion

Our work on the VisHive toolkit in this project is focused on distributing JavaScript code and computational tasks across multiple devices. Meanwhile, IPython [122] Notebooks—a web-based interactive shell for Python—are quickly becoming the main platform for scientific computing in the web browser. One of the reasons for the success of IPython for scientific computing is the immense ecosystem of Python packages available for all conceivable computational needs. Obviously, VisHive is not a replacement for IPython, but rather fills a niche that is very different from the greater mandate of IPython: integrating computation in a web-based visualization setting, which is already going to be JavaScript-based given the current state of visualization toolkits for the web. IPython, in contrast, is still speciality software that is not considered useful for the general population, is therefore not integrated with standard browser installations, and thus requires a separate download.

The same argument extends to general server-based, cloud-based, or cluster-based computational platforms. VisHive is **not** intended to replace such platforms, but instead provides an example solution for the common situation when a user has access to multiple local devices that could be formed into an ad-hoc cluster to help with computation performed in the browser running one of them. Since mobile devices as well as personal computers are exclusively designed for focused use—i.e., with one user using a single device, and not many devices at once—these additional devices are underutilized anyway. Our toolkit offers a lightweight approach to leverage these devices that is lightweight easily integrated with current web development practices.

Another aspect to note is that VisHive does not provide any explicit support for how to distribute computation so that it can be assigned into manageable chunks, sent off to separate cells, performed separately, and then recombined correctly by the master. Our focus in this work has been on the distributed computation mechanism itself, and not the distributed algorithms you would run on the individual cells. There exists vast amount of work in fields such as parallel computing, distributed systems, and high-performance computing that can begin to guide the design of suitable algorithms that can be run on top of VisHive.

VisHive is focusing specifically on computation and data splitting in visualization pipeline. It supports distributed computing through various stages and utilizes idle computational resources, especially mobile devices to aid data analysis. Connections through web browsers make it easy to use and resilient to node joining and leaving. VisHive shortens the time of data management and makes visualization rendering faster, which in turn is more responsive to user interactions during the data exploration. It provides a framework to package distributed computation for visualization easy and convenient.

While VisHive can provide advantages and convenience to performance in visualization without additional costs but a browser, there are some essential limitations for this framework that are stated below:

- **Framework:** Even if VisHive aims for supporting heavy computation in a distributed way, it is not a replacement of any existing framework that resolves the problem. The main issues are:
  - **Scalability:** VisHive is not well suited for deploying tasks to a large number of devices. Our approach utilizes the "nearby" available resources to aid computation that are otherwise often ignored. We have tested with up to 10 devices connected to the cluster, while due to the limitations of web browser, the interface of master device freezes when the number of devices are larger than 10. This also increases the overall processing time. From our experience, the optimal number of devices are between $3 - 6$, which aligns with the typical number of devices one person would have in the office.
  - **Data Sizes:** Since VisHive hosts all the data on the master and slaves and masters exchange tasks and data, the toolkit requires significant bandwidth for large data sizes.

- **Mobile Devices:** While we have illustrated a framework in this proposal for the advantages of using mobile devices to speed up computation, such devices are not always ideal for this purpose:
    - **Battery:** Battery life is a precious resource for most mobile devices. In fact, many mobile devices are designed to go to sleep if left inactive to conserve energy, which typically suspends JavaScript execution.
    - **Computational Resources:** Many mobile devices provide so limited computational resources so as not to be worthwhile to include in an ad-hoc cluster to contribute to a task. In a typical setting with VisHive(2 - 4 devices), the computation time of mobile devices are normally 3-4 times larger than a laptop. This may be due to the I/O constraints and less powerful CPU for computation purpose.
    - **Networks:** VisHive may trigger additional wireless network charges if an algorithm requires each participating device to download a duplicate of the dataset. On the other hand, as discussed in the evaluation section, data transfer over wireless networks take an inevitable portion (usually between 10% - 35&) of task processing time, which limit large scale computation tasks.

Besides, a NodeJS server is used in the current VisHive implementation for matchmaking purposes. While it is easy to connect cells and establish the hive, a server is not the optimal solution for matchmaking in many situations, especially for mobile devices. Typing in IP address is also slow and complicated, and sometimes raises security issues. One alternative way is to use Bluetooth or other near-field communication protocols. These protocols are more applicable for portable devices, however, they have more restrictions on data transfer speed and the distance range of connecting devices. Another way is that cells take a picture of QR code to join the hive. The prerequisite for this method to work is a camera and QR code identification application or mechanism on the device.

Nevertheless, we think VisHive outlines an exciting area for the future as the toolkit is easy to use without any additional packages installations, and computation and network connectivity becomes cheaper and cheaper. We also believe that VisHive can encourage other ideas from the field to better tackle these limitations.

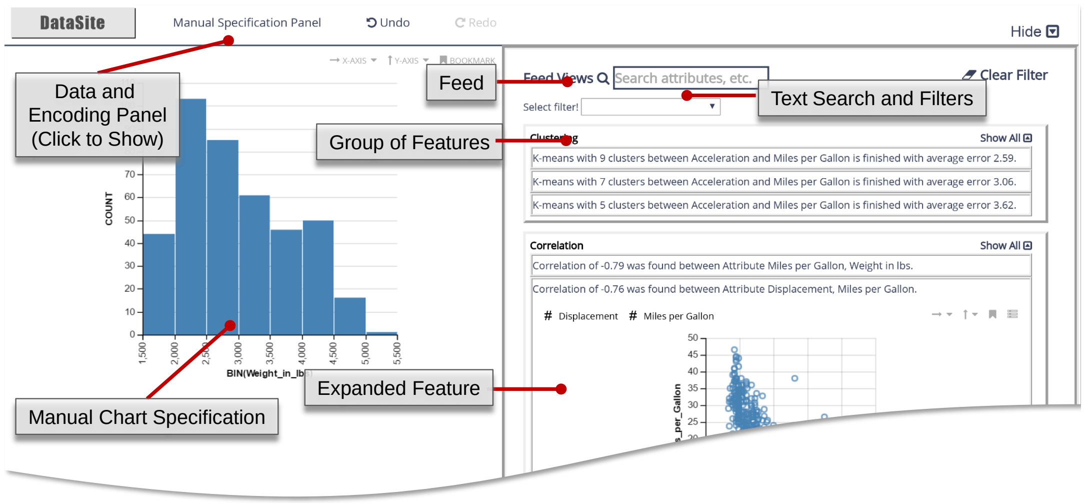
4.7 Conclusion

We have presented VisHive, a JavaScript toolkit that allows for connecting multiple devices into an ad-hoc cluster using just the web browser as the computational platform. Devices become cells in a hive where a master allocates and recombines jobs to slaves that perform the actual calculation. The communication between the cells is performed using direct browser-to-browser connections in a peer-to-peer architecture, thus requiring no central computation management server or connection to the Internet. The matchmaking service needs to reside on a server within the same local network to provide connections in the cluster. We briefly discussed the VisHive API and declaration of some functions for public access. To showcase the utility of the technique, we presented four example implementations of distributed algorithms, including a distributed web crawler with text analytics, an incremental database query, a density-based clustering algorithm, and a dimension reduction method. Our performance evaluations using these four applications show a significant speedup basically linear with the number of connected cells.

Chapter 5: Supporting Proactive Visual Exploration using Automatic Server-Side Computation

In this chapter [1], we present DataSite, a proactive visual analytics system where the user analyzes and visualizes the data while a computation engine simultaneously selects and executes appropriate automatic analyses on the data in the background (Figure 5.1). The underlying design rationale for DataSite is that CPU cycles are cheap, whereas human cognitive effort is not. By continuously running all conceivable computations on all combinations of data dimensions, ranked in order of perceived utility for the specific data, DataSite uses brute force to relieve the burden from the analyst of having to know all these analyses. Any potentially interesting trends unearthed by the computation engine are propagated as status notifications on a *feed view*, akin to posts on a social media feed such as Twitter or Facebook. We designed this feed view to support different stages of exploration. Status updates are continuously added to the feed as they become available during the exploration. To provide a quick overview, they are presented with a brief description that can be sorted, filtered, and queried. To get more details on an individual response without committing to the active path of exploration, we allow the analyst to expand an

[1]This chapter is an adaption of the paper [123] in Information Visualization Journal.



K-means with 9 clusters between Acceleration and Miles per Gallon is finished with average error 2.59.

K-means with 7 clusters between Acceleration and Miles per Gallon is finished with average error 3.06.

K-means with 5 clusters between Acceleration and Miles per Gallon is finished with average error 3.62.

Correlation of -0.79 was found between Attribute Miles per Gallon, Weight in lbs.

Correlation of -0.76 was found between Attribute Displacement, Miles per Gallon.

Figure 5.1: DataSite is a proactive visual analysis system that allows the analyst to explore data on the web-based client using a standard visualization interface (data, encoding, and manual chart specification panel), while a server-side component automatically selects and executes relevant computations without prompting. Features gleaned from these analyses are surfaced and ranked in a Feed View (right) on the client, similar to posts in a social media feed.

update to see details in natural language as well as an interactive thumbnail of a representative visualization. Finally, the user can commit to an update to bring it to the manual specification panel, allowing for manual exploration.

We first describe the design rational driving the structure of DataSite, and then introduce the DataSite system. To demonstrate the utility of the system, we present results from two user studies involving exploratory analysis of unknown data, one that compared DataSite to a Tableau-like visualization system (PoleStar [124]), and one that compared it to a partial-specification visualization recommendation system (Voyager 2 [70]). Using DataSite's feed, our participants derived richer, more complex, and subjectively insightful findings compared to when using PoleStar, or even Voyager 2's recommendation feed. This supports our hypothesis that a true proactive analytics platform such as DataSite can improve coverage and increase complexity of insights compared to reactive or partial-specification approaches. This also enhances the exploitable computational support for visual analysis. Beyond the DataSite system, our approach can be applied to other exploratory analysis tools to promote richer exploratory analysis, even for non-experts, analysts pressed for time, or analysts unfamiliar with a dataset before exploration.

5.1 Design Guidelines: Proactive Analytics

The core philosophy for proactive analytics is the following:

Human thinking is expensive, whereas computational resources are cheap.

Following this philosophy, a proactive approach to visual analytics should automatically run computations in the background and present its features to the analyst in an endeavor to reduce the analyst's cognitive effort during the sensemaking process. In essence, the solution is to use the brute force computational power of the computer to help balance out the equation between the human analyst and the computer tool. This leverages the respective strengths of each partner while complementing their weaknesses:

- Human analyst: The human operator driving the analysis.
  - *Strengths:* creativity, intuition, experience, deductive reasoning.
  - *Weaknesses:* limited short-term memory, computational power, knowledge, and perception.
- Computer analytics tool: The tool facilitating the analysis.

  - *Strengths:* significant memory and computational power; large library of algorithmic techniques.
  - *Weaknesses:* no or limited creativity, intuition, or deductive reasoning.

Based on these ideas, we derive the following design guidelines for our (and future) proactive visual analytics tools:

D1 **Offload computation from analyst to machine.** The analytical tool should be designed so as to offload as much as possible of the analysis from the user. Given our core philosophy, this means that the tool should never be idle in reactive mode waiting for the user to act. Instead, it should always be running tasks in the background, and start another task as soon as one finishes.

D2 **Present automated features incrementally with minimal interruption to the analyst.** Automatic features derived by the background computational processes must be propagated to the user, but the presentation of these features should be designed so as not to interrupt the user's cognitive processes needlessly. These features should be accumulated in a feed where they can be easily surveyed and viewed at the user's own initiative rather than in a blocking manner that requires action.

D3 **Reduce the knowledge barrier of human thinking.** Data analytics is a nascent discipline with rapidly evolving methods, many requiring the data to support specific assumptions or exhibit certain properties, so it is often difficult even for expert-level analysts to stay abreast of current practice [125]. This is another situation where timely proactive support can save analyst effort by investing CPU time: the tool can simply run every conceivable analytical method from a large library of such methods (ordered by perceived utility) and only present interesting trends.

D4 **Eliminate "cold-start" through exposing potentially relevant features of the data early during exploration.** A challenge related to the knowledge barrier is the so-called "cold-start problem" [15]; the fact that, when beginning analysis on a new dataset, it can be challenging to know how to get started because the data can be overwhelming and difficult to get a handle on. Again, this can be mitigated by not choosing but simply performing **all** applicable analyses from a library of such methods.

5.2 The DataSite System

Our web-based implementation of DataSite consists of an interactive web client interface for multidimensional data exploration as well as a server-side computation engine with a plugin system where new components can be integrated. The client interface is a shelf-based visualization design environment similar to Tableau (and based on Polestar [124] implementation). The server-side computation engine currently features common multidimensional components such as clustering, regression, correlation, dimension reduction, and inferential statistics, but can be further expanded depending on the type of data being loaded into DataSite. Each computational plugin implements a standardized interface for enumerating and ranking supported algorithms, running an analysis, and returning one or several status updates to the feed view. Computational tasks are run in a multithreaded, non-blocking fashion on the server, and use rudimentary scheduling based on their perceived utility for the specific data.

The DataSite system consists of (1) a user interface for proactive visual analytics containing components for visualization authoring along with a feed view, and (2) a proactive computation engine continuously running background modules on a target dataset (D1). The user interface (Figure 5.1) runs on a modern web browser and consists of a manual visualization view coupled with a *feed view*. In particular, the feed view accumulates features as status updates (D2) consisting of a textual description and a representative interactive visualization. Working in concert, the feed view reduces the knowledge barrier (D3) by continuously displaying trends from the proactive computation engine. The feed also provides a starting point, eliminating the cold start problem (D4).

5.2.1 Visualization Interface

The DataSite interface comprises a data schema panel, an encoding panel, a manual chart specification view, and a feed view (Figure 5.1). The data schema, encoding, and chart specification views together compose a basic shelf-based visualization system that the analyst can employ to explore the data in a conventional way, potentially disregarding the proactive analysis entirely. The feed view is the

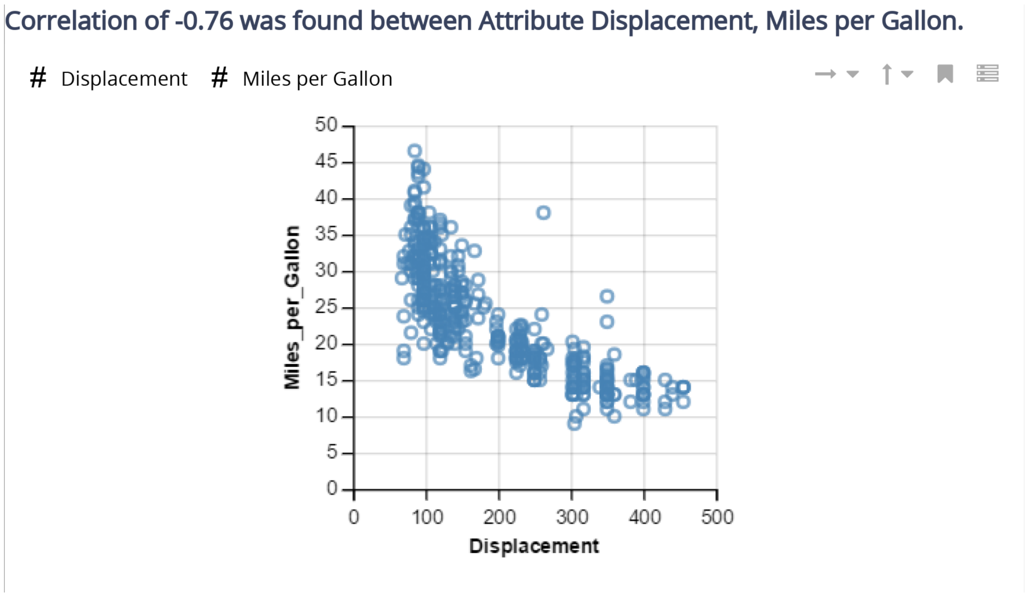


Figure 5.2: Example of features in the feed: a brief textual description ("Correlation metric between Miles per Gallon and Displacement attributes in a Cars dataset.") with a corresponding auto-generated chart (scatterplot for these two specific attributes).

key interface-level contribution of the DataSite system, and accumulates features generated by the computation engine. To give ample space for the analyst's navigation through the interface components, the feed is placed on the right of the manual specification view, and the manual shelf panels (data and encoding panel) can be hidden.

The feed view is inspired by social media feeds, where events posted by participants appear in a dynamically updating list. A *data feature* in the feed is a notification from a computation engine. The feed view can be searched and filtered, sorted by the computational measure, the time it was produced, or in simple alphabetical order, and grouped by their type. Each feature is initially represented as a textual description explaining the underlying computation task. Users can expand a feature to see more of the text as well as an associated chart for the data attributes processed by the underlying computation (Figure 5.2), or collapse it when needed. Here, we describe the textual description and charts within the feed view. One thing to note is that in this chapter we use “data attributes” and “data fields” interchangeably, representing attributes in the dataset.

- **Textual description:** Text that describes a feature presented on the feed view in a proactive manner. For example, for the Pearson correlation coefficients [126] between *Weights in lbs* and *Miles per Gallon* in cars dataset [127], the textual description is: “Correlation of 0.5 was found between attributes *Weights in lbs* and *Miles per Gallon*.” This active description gives the analyst the sense that the computer is their collaborator in helping them explore the data. To avoid overloading the feed with an excessive number of features, we combine related trends and illustrate them with a single chart (e.g., min/max are combined, described as a range, and shown on a bar chart, see Fig. 5.4).

- **Charts:** Manual view specification yields full control to the analysts, but may cause high cognitive load. To avoid this, DataSite shows the most efficient encodings for each chart corresponding to tasks from a computational module according to the existing metrics [58–60]. For instance, with two categorical attributes, DataSite renders a heatmap (Figure 5.3) with the frequency counts marked in color in each area of the intersections. Similar to the approach in previous research [12, 67], charts can be moved to the main view panel by clicking a *specify the chart* icon on the top right. Furthermore, charts highlight aspects of the underlying computation as visual cues for the user: for example,

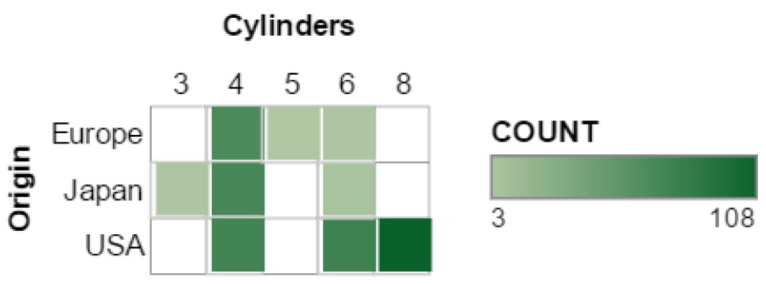


Figure 5.3: Representative chart (heatmap) automatically generated for co-occurrence frequency counts of two categorical data fields (origin country and number of cylinders) in a Cars dataset. Darker color indicates more counts in that category combination; in this example, eight-cylinder cars from the USA.

charts generated from the clustering computation will highlight the clusters within the chart in color.

At the same time, analysts can post an important view in the manual chart visualization window, which will save that view as a feature in the feed. The feed view keeps track of these user-generated features as a separate category of human computations. This is the same as bookmarking charts and in the future we plan to make the feed a collaborative space, where either human or computer post features to allow sharing of findings. Charts are lazily-rendered only when clicked, thus reducing the page load significantly.

5.2.2 Computation Engine

The DataSite computation engine begins analyzing a dataset as soon as it is uploaded. A scheduler will pass the dataset through its entire library of loaded computational modules, receiving an estimate of the computational complexity and relevance from each module based on the metadata—number of attributes, types, and dataset size. These two metrics will then be used by the scheduler to determine which modules to run, and in which order to run them. A single module can yield several tasks; for example, a simple Pearson correlation module would create a task for each combination of numerical data dimensions.

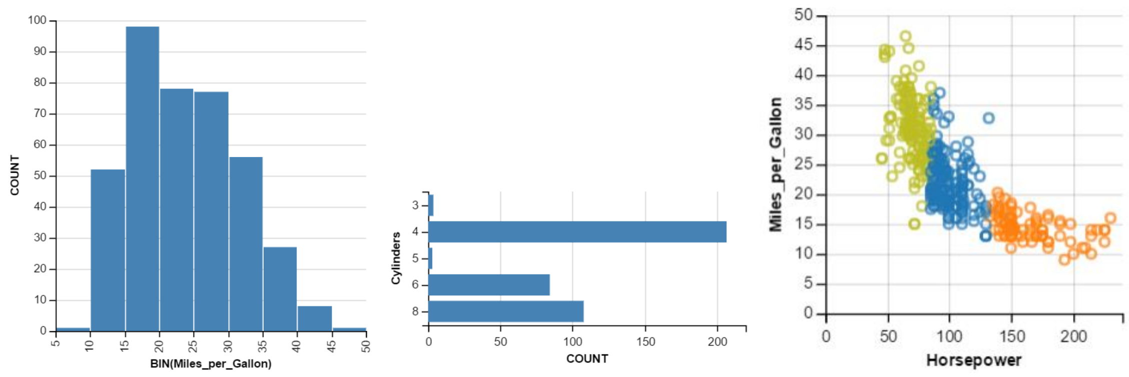


Figure 5.4: Example chart types for different computational modules used in DataSite. From left to right: histogram bar (mean/variance), histogram line (min/max), and scatterplot (clusters in 2D).

The scheduler is multi-threaded using a computational thread pool, executing each computation in the predetermined order. For each finished task, the computational module will generate a status update that will be pushed to the visualization interface. As soon as a computational thread is freed up, the scheduler will recycle the thread for a new task. In this way, the interface is never blocked by complex, long-running tasks. Furthermore, each computation module executes independently and is easily managed. A single module failing does not affect the overall system. In future work, we anticipate letting the user guide the computation order, either implicitly (by analyzing which data the user is interacting with), or explicitly (by providing specific interactions to guide the computation).

By virtue of this modular architecture, DataSite can be easily extended with new computation modules. The current implementation provides statistical analysis, clustering, and regression modules. Table 5.1 gives an overview of the modules implemented so far, whereas Figure 5.4 shows sample charts created in the feed view for some computation modules.

Modules	**Data Formats**	**#Attr.**	**Chart**	**Descriptions**
Mean/variance	numerical	1	histogram bar (Fig. 5.4)	Attribute A has mean of X with variance of Y.
Min/max (range)	numerical	1	histogram line	Range (min, max) was found in attribute A.
Freq. counts	categorical	1	aggregation (Fig. 5.4)	X was the most/least frequent subcategory in attribute A.
Freq. counts (comb.)	categorical	2	heatmap (Fig. 5.3)	Most frequent combination was found between X in attribute A, and Y in attribute B.
Correlation	numerical	2	scatterplot	Correlation of A was found between attribute X and attribute Y.
Clustering	numerical	2	scatterplot (Fig. 5.4)	Kmeans with N clusters between X and Y has average error E.
Regression	numerical	2	regression line	Linear Regression between X and Y has estimate error of E.

Table 5.1: Example computational modules with corresponding data and chart types. We have currently used algorithms working with one or two data attributes in our computation engine. Brief textual descriptions for each module are also listed.

5.2.3 Implementation

DataSite is based on a client/server architecture. The client side is developed using AngularJS,[2] a JavaScript-based web application framework. The visualization functionality in the DataSite client is based on the PoleStar interface (available as open source) [124], which is built on top of Vega-Lite [96].

We implemented the computational engine using Node.js,[3] a non-blocking server-side JavaScript framework. Datasets of interest can be uploaded by the user on the client interface, and sent to the server. The server processes them using the engine and proactively sends the finished features to the feed view. This structure enables managing a wide array of input data formats, and scales to large data-sets. In essence, the server does all the heavy lifting: loading data, maintaining the connections to clients, executing computational modules, and updating features.

5.3 Evaluation Overview

DataSite creates a new method for visual exploration through a mixture of manual and automated visualization specifications driven by proactive computations. For this reason, we are interested in understanding whether the exploratory analysis with DataSite supports bootstrap understanding and broad coverage of the data. At the same time, we are also curious about knowing how/why the feed helps, and how it changes the analyst's approach in finding features. To evaluate DataSite,

[2]`https://angularjs.org/`

[3]`https://nodejs.org/`

we conducted two user studies: (1) comparing with a manual visualization specification tool, PoleStar, focusing on data field coverage; and (2) comparing with a visualization recommendation system, Voyager 2 [70], focusing on data exploration to compare the effects of adding a *Feed* (in DataSite) versus *Related Views* (in Voyager 2). In other words, Study 1 aims to understand the fundamental utility of the feed view itself, while Study 2 expands this to understanding DataSite's proactive visual analytics workflow compared to a recent visual recommendation system.

5.3.1 Dataset

To enable comparisons of our results with Voyager and Voyager 2, we reused the same datasets for our studies. One is a collection of films ("movies") containing 3,178 records and 15 data fields, including 7 categorical, 1 temporal, and 8 quantitative attributes. The other dataset contains records of FAA wildlife airplane strikes ("birdstrikes"), which contains 10,000 records and 14 data fields, with 9 categorical, 1 temporal, and 4 quantitative attributes. These two datasets have similar complexity (w.r.t. number of attributes), and are easy to understand.

5.3.2 Study Design and Procedure

In both user studies, we used 2 tools with 2 datasets (one dataset on each tool interface). Participants in both studies started with an assigned tool and dataset, and then moved to the second interface. To deal with learning effects, we counterbalanced the order of tools and datasets—half of our subjects used PoleStar/Voyager

2 first and the other half used DataSite first (similarly with the dataset).

Each participant began a session by completing a short demographic survey and was then introduced to an assigned first interface. The participants were first shown the interface and a tutorial on how to use the tool with an automobile dataset [128] for training purposes. For DataSite, they were also shown the feed view and its associated operations. The participant was then allowed to train using the interface with the automobile dataset, and were encouraged to ask questions about the dataset and tools until they indicated that they were ready to proceed.

The experimenter then briefly introduced the participant to the experimental dataset and asked him/her to explore the dataset "as much as possible" (open-ended) within a given time of 20 minutes. They were asked to speak out aloud their thinking process and insights. We did not ask the participants to have specific questions to answer during the session, as this may bias them in exploration, and limit their focus to specific data subsets rather than the whole dataset. After completing a session with the first tool, the participants repeated the same procedure for the second tool and dataset. After completing the tasks for both tools, they were asked to complete a questionnaire with Likert-scale ratings on the efficiency and usefulness of each tool as well as the participant's rationale for their ratings. Participants were also encouraged to verbalize their reasons for ratings and their comments on the tools. Each session lasted 60 minutes.

All the sessions were held in a laboratory setting in our university campus. Both tools ran on Google Chrome web browser on a Windows 10 laptop with a 14-inch display. The experimenter observed each session and took notes. Participant's interactions with the tool were logged into files, including application events. The audio of the session was also recorded for further analysis.

5.4 User Study 1: Comparison with PoleStar

In this study, we compare DataSite with a Tableau-style visual analysis tool (PoleStar). As described earlier, this study was motivated by a fundamental question: what happens when you incorporate a feed view into a conventional visualization tool. We therefore studied the data field coverage during open-ended visual exploration influenced by the Feed in DataSite against Polestar (a baseline interface without the Feed view). Note that apart from the Feed view, the DataSite interface resembles the PoleStar interface. Our hypotheses were: (1) DataSite would have higher data field coverage and more charts viewed, (2) DataSite would allow exploration of complex charts with multiple encodings (capturing multiple attributes), and support faster understanding of the data.

5.4.1 Participants

We recruited 16 paid participants (7 female, 9 male) from the general student population at our university. Participants were 18 to 35 years of age, with some prior data analysis and visualization experience. All of them have experience with data analysis and visualization tools: All (16) had used Excel, 10 had used Tableau, 7 Python/matplotlib, 7 R/ggplot, and 3 had used other tools (such as SAP business tools). No participant had previously seen or analyzed the datasets used in our study. They had not heard of or used DataSite or PoleStar, though some found the PoleStar interface to be similar to Tableau.

5.4.2 Results and Observations

We used the linear mixed-effects model [129, 130] for our analysis of the collected usage data. We modeled the participants and datasets as random effects with intercept terms (per-dataset and per-participant bias), and regarded different tools and the order of tool usage as fixed effects. This setting accounts for the variance of tools and datasets with individual subject's performance during the study. We used likelihood-ratio tests to compare the full model with other models to evaluate the significance of difference.

To assess the broad coverage of data fields, we consider the number of unique data field sets. Users may have been exposed to a large number of visualization charts, while the unique field sets shown and interacted with are conservative and reasonable measures of overall dataset coverage. Based on this, there is a significant improvement of data attribute coverage with DataSite (30% increase compared to PoleStar: $\chi^2(1) = 19.26$, $p < 0.005$). Participants interacted with more charts, both from the feed as well as by modifying encodings from the charts present within the feed. This confirms the first hypothesis.

There are more multi-attribute charts (encoding two or more data attributes) that participants viewed and interacted with using DataSite than PoleStar ($\chi^2(1) = 10.31$, $p < 0.005$). This is expected since DataSite provides pre-computed features, while participants had to manually create all visualization charts themselves in PoleStar. 75% participants have seen at least 50% more data fields in DataSite. Participants also found twice the number of charts using DataSite that are informative and worth "speaking out" ($\chi^2(1) = 7.82$, $p < 0.005$). 10 participants have created more than 3 advanced charts with the help of feed (and "spoke out" about them): they started with charts from feed and added more data fields as encodings to the charts. This suggests that the DataSite system through its Feed view leads to the users viewing more number of charts that are beneficial from their perspective. It also indicates that DataSite encourages the user to reach complex (multi-attribute) charts during visual exploration. This confirms our second hypothesis.

Participants showed a great interest in the features within the feed view. Most of them spent at least 25% of time on exploring the feed itself. All participants felt that the feed is useful for analysis and provides guidance of "where to look at" in the data. They rated DataSite higher than PoleStar in terms of efficiency (Likert scale, 1 to 5, mean: 4.67 vs 3.40) and comprehensiveness (mean: 4.20 vs 3.21). All participants rated 3 or higher (out of 5) for the usefulness of the feed.

5.5 User Study 2: Comparison with Voyager 2

The results from the first study were promising and they answer our fundamental questions about the utility of the DataSite feed view. In Study 2, we compared DataSite with Voyager 2, a modern visualization recommendation system. The goal was to observe differences and further understand the utility of the feed in DataSite compared to the Related Views and Wildcards in Voyager 2.

5.5.1 Participants

We recruited 12 participants (8 female, 4 male) from our university. All had similar demographics (between 18 and 35 years of age) and data analysis experience as before: all participants (12) had used Excel, 8 Tableau, 6 Python/matplotlib, 1 with R/ggplot. They had not heard of DataSite or Voyager 2, or seen the datasets being used in our study.

5.5.2 Hypotheses

Our hypotheses for Study 2 are, (1) DataSite will provide comparable if not more data field coverage owing to its rigorous computation engine; and (2) DataSite will better guide the user's exploration towards faster and comprehensive understanding in the given time.

5.5.3 Results: Quantitative

We used the same linear mixed-effect model for statistical analysis in Study 2 similar to Study 1.

5.5.3.1 Data Field Coverage

We first looked into the participants' performance separately for both datasets (movies and birdstrikes), and compared the effects of visualization tools. We consider the number of unique field sets that users have shown and examined, respectively (similar to the previous study). In Figure 5.5, we see that for movies and birdstrikes datasets, the number of unique field sets that users interacted with (hovered mouse for more than three seconds) is similar: DataSite has 5 and 4 more unique field sets respectively in the birdstrike dataset (median: 30 in DataSite vs. 25 in Voyager 2) and movies dataset (median: 31 in DataSite vs. 27 in Voyager 2). Overall, DataSite promotes slightly more data field coverage in total (mean: 30 and 26), mainly because the feed contains an exhaustive list of features across computational modules.

In regard to the number of unique field sets that have been shown (user may look through the charts without interaction) to the users, DataSite users ($mean =$ 43, $s.d. = 19.7$) were shown fewer charts than Voyager 2 ($mean = 54$, $s.d. = 13.5$). The reason may be that Voyager 2 shows charts by default, while DataSite needs user interaction to expand the features in the feed to see the charts. As for the number of charts that participants spoke out aloud during the study, the tools have a significant difference ($\chi^2(1) = 7.34$, $p < 0.1$): DataSite ($mean = 14.53$, $s.d. = 2.04$) gave participants 30% more charts to "speak out" about, compared to Voyager 2 ($mean = 11.63$, $s.d. = 2.32$). In other words, participants found more charts to be informative and worth talking about using DataSite. Among all the

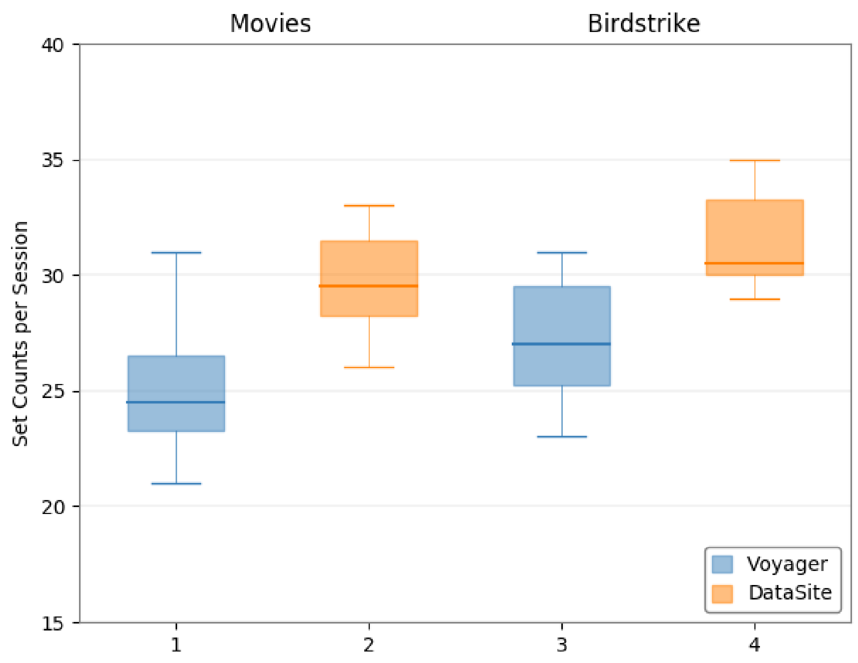


Figure 5.5: Box plot showing the distributions of number of unique field sets (that users interacted with) for the tools used on different datasets. DataSite has slightly larger number of unique field sets in both cases.

"speak out" charts, an average of 35% are directly from feed. Other "speak out" charts in DataSite are either moved from the feed to the main view and then edited, or manually created. This indicates that the feed view contributes to more data field coverage and more charts that analysts find useful and worth pointing out.

When using DataSite, all participants had viewed and interacted with the charts in the feed. Most of them (11/12) spent more than 30% percent of time exploring the feed. Two participants even used the feed as the main interface for exploring the datasets. Beyond this, two participants had interacted with more than 70% of total charts, and 75% of their "speak out" charts were directly from the feed.

5.5.3.2 Text Search and Filter Usage

We analyzed the usage of filters and text search bar. We were interested in observing whether filters and text search can aid them in searching for desired features within the feed view, and whether it is efficient and easy to use compared to *Related Views* and *Wildcards* in Voyager 2. All participants have used the drop-down filters at least 5 times, and 9/12 tried text search. 8/12 of them said that the filters and the text search were useful for quick search of the feed during the study session. 7/12 had used the combinations of text search and filter. Three participants found *Wildcards* in Voyager 2 to be not very intuitive. They used *Wildcards* fewer times during the exploration, which matches the results from Wongsuphasawat et al. [70]. In comparison, filters and search options not only contribute to fast data exploration, but also improve the efficiency of drilling down into features during proactive visual analytics. We believe that this is one of the advantages of providing descriptions for the features shown in the Feed view.

5.5.3.3 User Ratings

We collected user's feedback and ratings for tools in the post-study survey. For each tool, participants were asked to evaluate the technique based on the efficiency, enjoyability, and ease of use, on Likert scale ratings from 1 (least) to 5 (most). The participants rated DataSite ($\mu = 4.32$, $\sigma = 0.67$) higher than Voyager 2 ($\mu = 3.92$, $\sigma = 0.67$) regarding the efficiency. For enjoyability and ease of use, the ratings are comparable: enjoyability (DataSite: $\mu = 4.33$, $\sigma = 0.65$; Voyager 2: $\mu = 4.08$, $\sigma =$

0.67), ease of use (DataSite: $\mu = 3.92$, $\sigma = 0.85$; Voyager 2: $\mu = 4$, $\sigma = 0.60$). When asked about the comprehensiveness of their explorations of the dataset, 7/12 users rated DataSite higher and 4/12 rated both tools with the highest (5) score. Two participants gave lower ratings for DataSite compared to Voyager 2 and mentioned that it is because they felt in Voyager 2 it was easier to browse multiple charts while in DataSite they had to explicitly click. Overall, DataSite was seen to be more efficient and presenting a more comprehensive coverage of the data fields with respect to visual exploration than Voyager 2, while maintaining the similar level of enjoyability and ease of use.

Users also responded very positively when asked whether features in the feed provide guidance in their data analysis: 50% chose 5 and the rest chose a 4 rating. When it comes to comparison (Fig. 5.6) between two tools on a 5-level symmetric scale (with range $(-2, 2)$.), most participants (11/12) preferred DataSite ($\mu = 1.25$, $\sigma = 0.87$) to be most useful or useful for data exploration. Beyond this, participants were asked about their preferences between the two tools for focused question answering (as questioned by Wongsuphasawat et al. [70]). 7/12 users preferred DataSite, and 4 were neutral with no preference, with 1 preferring Voyager 2 (rated -1). This is a little surprising since DataSite was primarily designed for visual exploration (and not question answering).

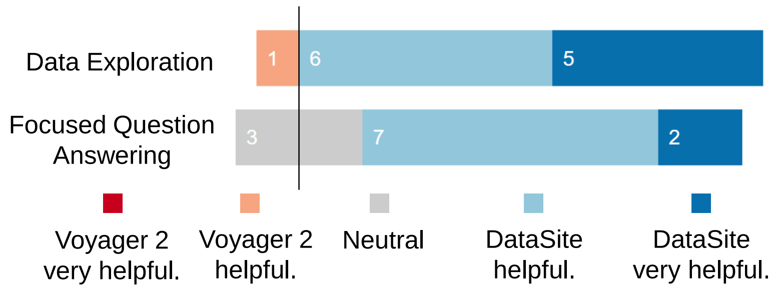


Figure 5.6: Subjective ratings of user preference in terms of the visualization tools for open-ended exploration and focused question answering. DataSite received higher preference in both open-ended exploration and focused question answering. 11/12 participants prefer DataSite for data exploration, and 9/12 prefer DataSite for focused question answering.

5.5.4 Results: Qualitative

To better understand the results from the statistical analysis, the participant ratings, and how DataSite helped participants explore the datasets, we present our observations below.

5.5.4.1 When the Participants used the Feed

The 12 participants were divided evenly to have different orders of the tools (DataSite first or Voyager 2 first). Four out of six who used Voyager 2 first, examined the feed (first interacted with the feed) in the beginning of their analysis with DataSite. For those exposed to DataSite first, 5/6 did the same. The rest started their manipulation first with manual specifications. It is worth noting that when the participants did not have any idea of how to construct interesting charts to get insights, they (8/12) switched to the feed for charts and inspirations (during the middle 10 minutes). 10/12 scanned through the feed at least once in the last 5 minutes of the session. 9/12 participants returned to the feed at least 3 times during the study. All of them specified at least 3 charts from the feed into the main view. This suggests that the feed can help analyst in multiple phases of exploration.

5.5.4.2 In-depth Data Exploration

Users usually create charts in manual specification tools with less than three attributes for encodings to limit the information encoded to a perceivable level. 7/12 found more advanced charts (3 or more data fields/attributes, the same below) that they "spoke out" in DataSite than Voyager 2 (at least 20% more). They mentioned that the summary in feed provides descriptive analysis, while charts alone in Voyager 2 may need more time to understand. It is worth noting that one participant used feed as the only interface for data exploration without additional manual specifications, and none did the same in Voyager 2. She explained that the feed provides a systematic approach towards analyzing the dataset, while she had difficulty understanding *Related Views* in Voyager 2.

# Charts	**Simp. Stats**	**Corr**	**Freq**	**Clust**	**Regr**
mean	2.25	4.38	4.31	3.54	3.26
std. dev.	1.25	2.5	3.46	1.02	1.57

Table 5.2: Statistics (mean, s.d.) of the number of charts from different computational modules that participants talked about during the study. Participants interacted with advanced features more (e.g., correlations, frequency counts, clustering, etc), while few features regarding simple statistics (min/max and mean/variance) were examined.

5.5.4.3 "Speak out" Charts in the Feed

The number of "speak out" charts that users verbally referred to during the study revealed interesting aspects for data analysis by general users. Table 5.2 gives mean and variance of features in different categories that the participants "speak out" about. Participants were more interested in plots of multiple numerical fields and categorical fields, rather than a single numerical field. Specifically, they merely viewed the charts in range/mean/variance modules (average number of charts are around 1), and from our observations, they skimmed through the natural language descriptions but did not click to see the charts. This implies that simple statistics are not interesting enough for analysts to examine, or the text descriptions alone are sufficient to understand.

For complex computations (correlation and clustering), charts are viewed more by expanding their textual description in the Feed. This is because there are usually no intuitive attribute combinations to creating informative charts with data fields (participants had to rely on random combinations or based on their general understanding). After seeing the charts in the feed, they all agreed that those charts were more informative than the ones they created by manual specification. This motivates us to choose other suitable modules to make the feed more fascinating and user friendly to explore.

5.5.4.4 Inspirations from the Feed

The feed view provides recommendations for visual data exploration from an analytical perspective. The features suggest certain combinations that yield effective visualizations. All the participants manually specified similar charts (w.r.t. encodings) after they had seen the charts within the feed, especially heatmaps representing frequency combination of two categorical fields. More than 80% (10/12) of the participants mentioned that the feed gave them some ideas of which features and encodings can be used to make the chart more informative. On the other hand, *Related Views* in Voyager 2 show visualization recommendations to users that can be easily browsed, but participants thought of them just as related charts rather than specific analytical insights. They browsed through *Related Views* a lot but had never considered about how and why the specific chart was suggested. Also, 2 participants felt that the descriptions sometimes were not very easy to understand.

5.5.5 Participant Feedback

In this section, we list specific comments, suggestions, and feedback from the free text comments in the post-study survey and audio recording transcripts. For example, participants described that DataSite helped the visual data exploration process: *"The feed helps gear you in the right direction, especially if you are new to a dataset. It tells you something notable that is worth looking into."* As for comparisons to Voyager 2, *"DataSite is more specific because it gives you the options with various kinds of results. The feed is very helpful in data analysis."* One participant even remarked that *"[DataSite] will be very useful for day-to-day usage, especially for advanced data analysis, and can be used in industrial applications."*

Participants also "spoke out" their findings, one said, *"For distributors, most of them [have movies of] 4 to 8 in IMDb rating."* Another example is, *"Most of the [birdstrike] accidents happened during the day time."*

Overall, the feed view was lauded, with one participant noting that *"the feed in DataSite provides a good starting point to visualize data if you don't have any idea about the dataset."* However, participants also provided suggestions on how to improve the feed. Said one participant, *"it would be better to make feed more user friendly, such as drag-and-drop to move charts into the main view."* The feed was also perceived to be daunting, or as one participant put it: *"the feed is very useful, but sometimes it has a lot of results and can be a little overwhelming."* Another participant compared the feed to Voyager 2, saying that *"in DataSite it is a bit difficult for me to understand the results in the feed, while Voyager 2 provides intuitive charts."* One participant suggested that *"it would be interesting if there were guided tips that can help when I'm stuck in a chart, such as 'try changing x and y axis' when the axis label is difficult to read."*

5.6 Discussion

Our results have shown that the feed interface expedites the process of data exploration both in breadth and depth. Compared with the study results in Voyager 2, DataSite has a comparable unique field set coverage. The reason why DataSite does not improve the coverage significantly is that Voyager 2 shows all the charts by default, while DataSite only shows charts on demand when participants click on the descriptions. In other words, DataSite requires participants to actively examine the charts in the feed rather than merely browsing them in Voyager 2's *Related Views*. Most participants preferred DataSite for data exploration, and rated the feed very useful to aid data analysis and provide trends and guidance of creating meaningful visualization. It is worth noting that DataSite also yielded higher ratings in focused question answering. While DataSite is not designed primarily for targeted exploration, the study reveals a potential effect on focused question answering. This also motivates us to consider what and how a targeted data analysis system should adjust, and what evaluations can be done to achieve that purpose.

One observation from our evaluation studies is that simple statistics (average, range, variance, etc) did not interest participants much. A comprehensive evaluation of what features would be more interesting to the analysts is needed. The salient features lower barrier for bootstrapping exploration. However, too many features may distract user's interest. We have to balance these carefully. While Voyager 2 also provides efficient visualization recommendations, results from our evaluation indicate that participants felt that the feed was more targeted and worth analyzing. Three participants noted out that while they were going through Voyager's related views, they sometimes forgot what they had seen using manual view specifications. We speculate that the fact that DataSite explicitly labels the features using a textual description facilitates more targeted analysis.

It is worth noting that DataSite exhaustively applies computations to all the possible data fields (and combinations). While this enhances data coverage, not all modules and corresponding charts represent a clear insight. For example, categorical attributes such as "name" may have thousands of entries, and it is very difficult to find salient trends via such a chart. While DataSite modules rank features by their significance, a more precise saliency measure is needed. The challenge is how to measure the efficiency of analytical features from a human perspective, and how to unify the metrics across various types of computations. This requires comprehensively measuring the efficiency for each visualization. This is further complicated by the fact that different analysts may have different perspectives, or the same analyst may have different perspectives depending on the question under study. For the automobile dataset, buyers may wish to see which car is more economic and safer (higher *Miles per Gallon* and fewer accident records), while sellers may be interested in popularity (higher profits and larger number of sales). These contexts should also be considered for customization and personalization of features. Automatic guided tooltip, suggested in one participant's comments, would be one way to achieve this.

5.7 Conclusion

In this chapter, we have presented DataSite, a visual analytics system that integrates automatic computation with manual visualization exploration. DataSite introduces the feed, a list of dynamically updated notifications arising from a server-side computation engine that continually runs suitable analyses on the dataset. The feed stimulates the analyst's sensemaking through brief descriptions of computational modules along with corresponding charts. Filters and text search bar enable quick scan and fast data exploration. Two controlled user studies evaluate the approach compared to PoleStar and Voyager 2, respectively, and show that significant performance improvements over the manual view specification tool (PoleStar) in both breadth and depth for data coverage, as well as useful guidance in exploration. It also provides more meaningful charts and features to analysts over Voyager 2, while maintaining similar ease of usage. The results are promising and indicate that the system promotes data analysis in all stages of exploration.

Chapter 6: Applying Proactive Visual Analytics to Genomic Domains

Integrative analysis of genomic data that includes statistical and computational methods in combination with visual exploration has gained widespread adoption. Many existing methods involve a combination of tools and resources: user interfaces (most commonly web browsers) that provide visualization of large genomic datasets, and computational environments (usually servers) that focus on data filtering, transformations and analyses over various subsets of a given dataset. While effective use of data analysis tools, like Epiviz, usually places the burden of steering data analysis on the user, specifically, exploring and testing possible hypothesis underlying the dataset, which delays their ability to interpret and follow up on analysis results based on their subject expertise and knowledge of specific data. In practice, existing biological data exploration tools such as Epiviz and Metaviz [131], combines computational genomic and metagenomic data analysis with interactive visualizations, and it requires the analyst to guide the analysis. In this chapter, we present the Epiviz Feed application, integrated with Epiviz [132], which combines proactive statistical analysis of genomic sequencing data with interactive visualization and exploration of various features.

6.1 Motivation Scenario

As a motivating example, We peform an integrative analysis of DNA methylation and gene expression across multiple cancer types [133]. Loss of DNA methylation in large partially methylated regions is recognized as a common occurrence in solid tumors. Changes in gene expression within these regions are also observed in solid tumors. The goal of this experiment was to identify the extent to which these large regions of methylation loss overlap across four different solid tumor types (lung, breast, colon, and thyroid) and the extent of common differences in gene expression.

We further move the focus of the analyses for this chapter to address the following use cases in an integrative analysis of this type. Suppose a data analyst has collected DNA methylation data across multiple tissues, for case and control populations (e.g., tumor and corresponding normal tissue), along with gene expression data. Also assume the analyst has identified large regions of differential methylation in cancer (in this chapter we use the minfi analysis package [134] for this purpose) based on the DNA methylation measurements. The workflow we use as a design principle is the following: the analyst chooses a genomic region of interest, e.g., a specific gene of interest, and the proactive analysis system would address the following questions:

- Do regions of differential methylation overlap for some pairs of tumor types?
- Are there significant correlation between methylation measurements in this

region across (normal) tissue types, how about cancer types?

- Are there significant differences in gene expression across normal tissue types? Across cancer types? Between normal and tumor in a specific tissue type?

- Are there significant correlations in gene expression between normal tissue types? Or between cancer types?

- Is there a significant correlation between DNA methylation in a gene's promoter region and its expression in a given tissue?

When results of the appropriate statistical inferences required to address the above questions become available, the analyst can then inspect the data leading to these inferences using the interactive visualization capabilities available through Epiviz. In the following sections, we describe our system design to support this use case.

6.2 System Design

We present the design and architecture of integrating proactive visual and statistical analysis with Epiviz software package. This web application includes browser based user interface, database support, and computational server. Figure 6.1 gives the overview of the three components: database, computational server, and front end application. While existing Epiviz application has combined computational environments with visualizations, it does not support the proactive functionality of analyzing genomic dataset. In this section we describe the attempt to take one step further to improve the computational server and visual interfaces within Epiviz to provide automatic and interactive features of the application. The key motivation, similar as DataSite, is to leverage CPU power to process the dataset and provide automatic statistical results to the analysts, thus reducing their workload.

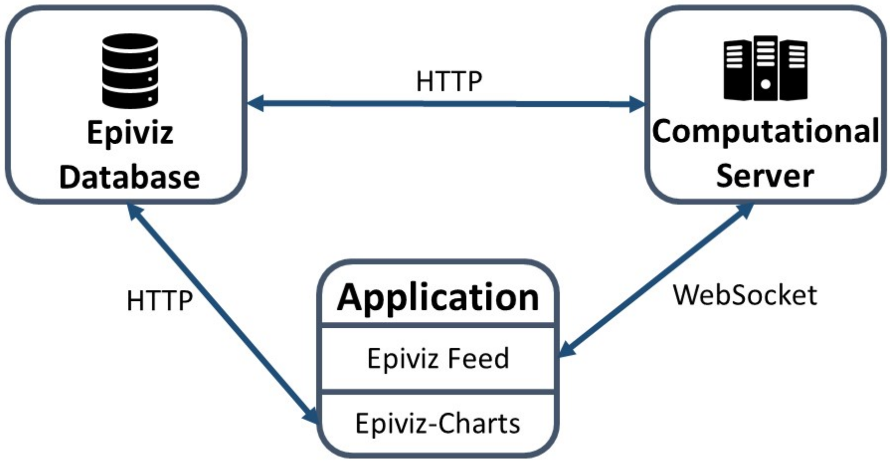


Figure 6.1: **Architecture of the proactive Epiviz framework**. The application works both as a genome browser and can be used to view the results from the computational server. If the analyst is using Epiviz Feed as a genome data browser, the application queries and visualizes data directly from the epiviz database. When the analyst navigates on the epiviz workspace, statistical methods are automatically computed on the computational server and the application has a persistent connection (using WebSockets) with the server to stream the results back to the Feed interface. The computational server also queries the epiviz database to perform analysis.

The main design guidelines underlying this work, that combine interactive visualization as provided by Epiviz and proactive statistical analysis are as follows:

1. Analyze the genomic data automatically and efficiently; and

2. Reduce the workload and knowledge barrier of human effort.

Epiviz provides rich visualization features and interactions, while efficient analysis requires substantial time and domain knowledge from the analyst. A proactive data analysis approach starts a series of analysis automatically and propagate results to the user dynamically when it finishes. Specifically, we take the idea from social media posts to provide statistical analysis results in a feed, which is convenient for analysts to explore.

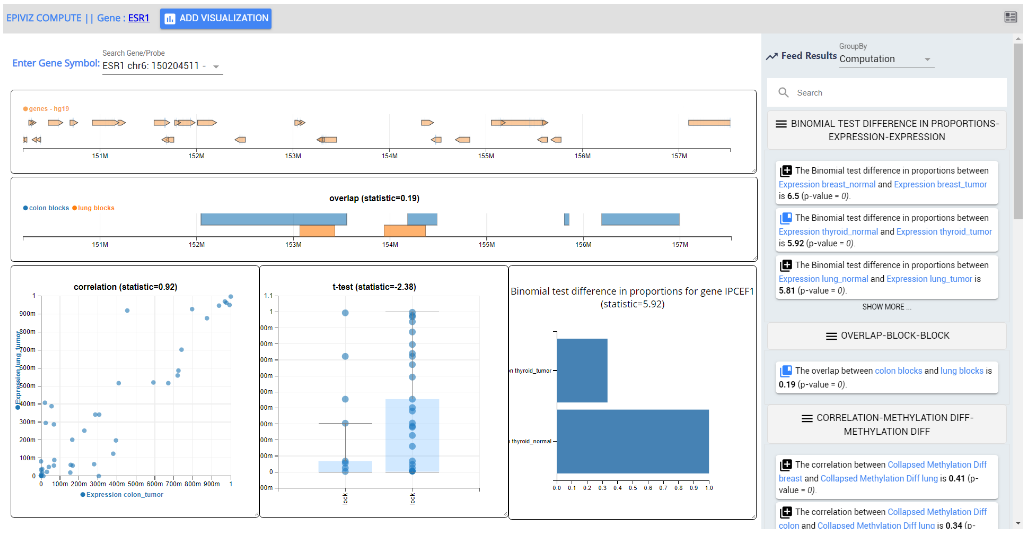
Visualization components are built using the Epiviz component library [135] on top of D3.js [19]. The user interface with feed uses Polymer [1], a JavaScript framework from Google. The existing Epiviz back-end handles genomic data storage and queries. Statistical analysis operations were implemented in Python using Numpy [136] and Scipy [137].

6.2.1 Visualization Interface

The Epiviz Feed user interface consists of an Epiviz navigation panel, as well as the result feed, which contains results from various statistical tests. The feed is updated dynamically as analyses are computed on the computation server. The feed is visible on the right side of the user interface, both to explore data independent of the feed results and can be collapsed.

As mentioned before, the feed interface is motivated by social media feeds, where posts from participants show up in a list. For a given genomic region, each result item is a type of statistical analysis on a subset of data based on the condition defined by annotations. In our motivating example, these are sample attributes, e.g, tumor or normal tissue. Each result from the computational server is converted into text in the feed that summarizes the statistical test. The feed can be searched,

[1] `https://www.polymer-project.org/`



Feed Results GroupBy Computation

Search

BINOMIAL TEST DIFFERENCE IN PROPORTIONS-EXPRESSION-EXPRESSION

The Binomial test difference in proportions between Expression breast_normal and Expression breast_tumor is **6.5** (p-value = *0*).

The Binomial test difference in proportions between Expression thyroid_normal and Expression thyroid_tumor is **5.92** (p-value = *0*).

The Binomial test difference in proportions between Expression lung_normal and Expression lung_tumor is **5.81** (p-value = *0*).

SHOW MORE ...

OVERLAP-BLOCK-BLOCK

The overlap between colon blocks and lung blocks is **0.19** (p-value = *0*).

CORRELATION-METHYLATION DIFF-METHYLATION DIFF

The correlation between Collapsed Methylation Diff breast and Collapsed Methylation Diff lung is **0.41** (p-value = *0*).

The correlation between Collapsed Methylation Diff colon and Collapsed Methylation Diff lung is **0.34** (p-

Figure 6.2: **Epiviz Feed proactive statistical analysis and interactive visualization of human gene sequence study.** The current genomic region has $startsequence = 3947153$ and $endsequence = 7164991$ within chromosome 11. On the left is an Epiviz Feed workspace visualizing genomic data from this region as tracks or plots. On the right, the feed lists all the automatic statistical results computed in this genomic region. It can be collapsed if needed. "GroupBy" can group the results by *Computation Type* or *Data Type*. The feed in this figure illustrates a groupby *Computation Type* mode, which has three categories: "OVERLAP", "CORRELATION", and "T-TEST". The search bar inside the feed provides text search and fast lookup through the results. For different statistical methods run on the server, the feed provides the result with a bried natural language description. The analysts can click on a result in the feed to quickly verify or visualize the corresponding data used to perform the statistical test. If the chart is already added to the workspace, the button before the feed text will change to blue. The feed text also highlights measurements used for the statistical test and can be clicked to filter other tests performed on this measurement.

filtered, and grouped by analysis type. Analysts can also filter the results by clicking on the measurement name (highlighted in blue) in individual feed items similar to hashtags in a twitter feed. This will then filter all results and show only those that inlcude the measurement in the statistical test. To interpret the result from the statistical test, the user can click on an item in the feed and instantly visualize the underlying data. The analyst may also click on the title of the visualization charts, which will scroll the feed to the corresponding natural language descriptions.

Fig. 6.2 shows the visualization interface of the proposed Epiviz Feed. The feed that contains statistical results computed by the server is shown on the right while Epiviz navigation panel on the left. The feed can be collapsed if needed. All the visualization charts will be shown in the workspace of the navigation panel. Users can add a chart by selecting the "Add Visualization" button on the navigation panel or by clicking on an item in the feed, which will add a chart, that illustrate the underlying statistic based on our experience. For example, a scatter plot for correlation tests and a bar plot for t-tests. We will present key features and components within the feed in the following section.

6.2.1.1 Epiviz Feed Stream

Each item in the feed represents a statistical test completed by the comptutational server. Each statistical test result is transcribed to a short sentence describing the underlying analysis and corresponding result. The pertinent data attribute names are *highlighted in blue* and analysis results are **bold** to make it identifiable.

The analyst can add the corresponding chart by clicking on the icon preceding the description. This will add a corresponding visualization chart in the workspace. The color of preceding icon will change from black to blue (bookmarked). Since every feed item is linked to a visualization, clicking a feed item multiple times will scroll the workspace to the linked chart. By default, the feed is organized by computation type, and ordered based on the type of analysis and significance of the statistical test. If the genomic region changes, the feed gets updated with the analyses for the new region.

In order to progressively update analysis results and keep the analyst informed, the server-side computations are batched and the results are propagated to the user using WebSocket connections [138]. The reason to use WebSocket instead of HTTP request [139] is to be able to stream multiple results and establish a persistent connection between the client and the server. Figure 6.3 shows the procedure of streaming between a client and server. Once the application is initially loaded or there is a change in the focused gene sequence region, the front end opens a web socket connection, sends a request to the server, and waits for the responses back. Once all the responses have been received, the server sends another message marking the end of the request. This WebSocket connection will then be closed. This "open on usage" ensures connection resources are efficiently recycled and will be wasted if the analyst is using the tool but does not trigger a request. To avoid filling the feed with an excessive number of results, we limit only statistically significant (generally we use $p - value < 0.01$) results to the analyst.

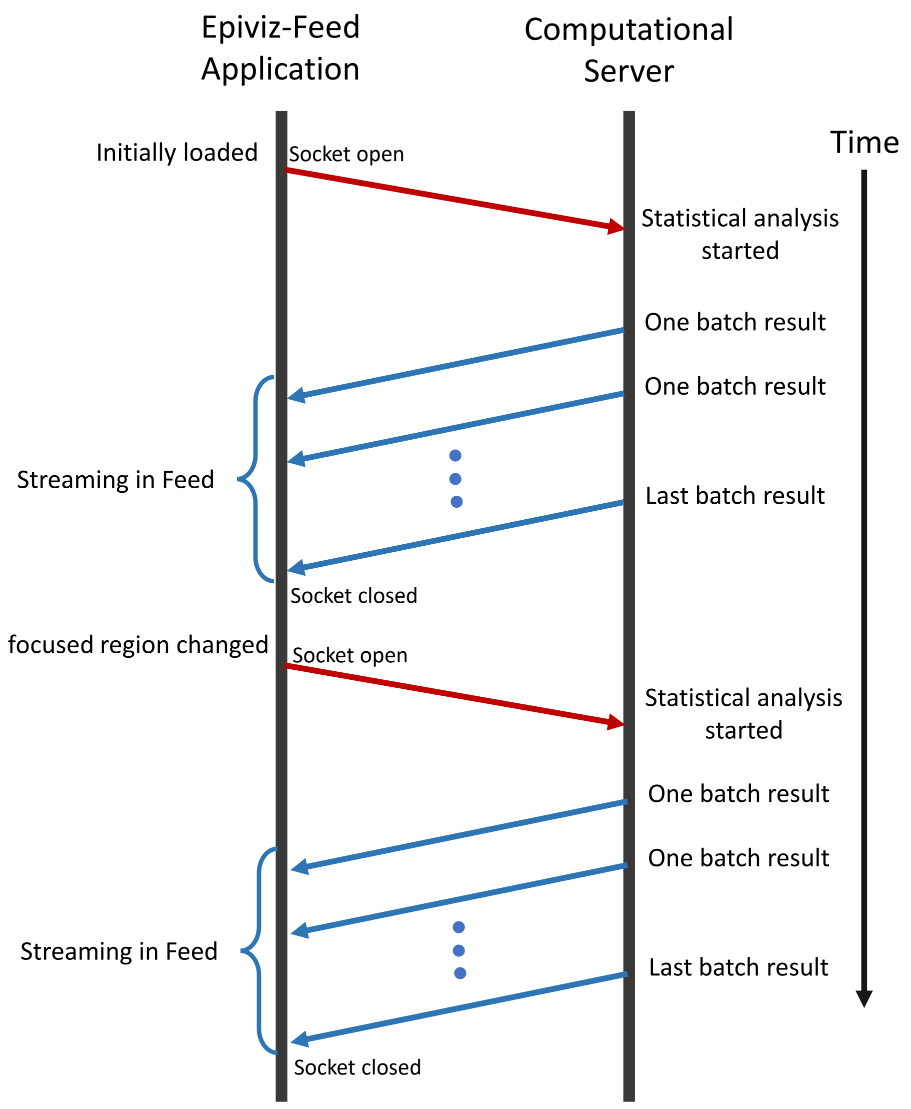


Figure 6.3: Statistical analysis results streaming mechanism using WebSocket. Whenever the application interface is initially loaded or the focused region changes, a WebSocket connection is established between the client and the server. Requests will be sent from the client to the computational server. Server analyzes and sends back the results whenever finished. When the last result is sent, the client closes the WebSocket connection and releases the communication resource for future or other client usage.

6.2.1.2 Plots and Charts of the Analyzing Results

As mentioned before, Epiviz Feed retains most features from Epiviz, which supports various visualizations of functional genomic data: plots such as heatmaps or scatterplots that visualize gene expression data across tissues, and line track or blocks track to visualize methylation signal data and peaks. The analyst can manually add a chart of various tracks to the workspace via the “Add Visualization” button on the top of the navigation panel. This is “visualization from data” approach, which depends more on the prior knowledge of the data. Alternatively, when the analyst looks through the feed and finds a statistical results interesting, one could click on the feed item to add the corresponding plot, which is “visualization from analysis”. This gives the analyst an opportunity to explore the statistical features of the measurements before adding and viewing charts that may not be of interest. It is easier and more convenient to quickly go through list of analysis results, especially for analysts who are willing to find underlying statistical properties. Besides, “visualization from analysis” procedure only renders charts during data exploration when user requires (lazy rendering), thus reduces page loads of web browsers.

6.2.1.3 Text Search, Filters, and Grouping

The feed incorporates text-based search (search box on the top of the feed) for quick navigation through text descriptions of the analysis results. The analyst can type in data attribute name, analysis type, or the value of the results to search and filter the required features in the feed. When the genomic region of interest changes (i.e., the region specified in navigation panel), existing visualizations added in the workspace updates, and the feed will also update accordingly with analysis of the data in the new region.

Besides customized text search, a list of pre-defined grouping methods are also available in the dropdown menu at the top. When the analyst selects one method, all the results in the feed will be grouped based on the method (e.g., “computation type”), each group of analysis results will be a sub-list under that category. For example, method “computation type” has “correlation”, “t-test”, and “overlap” (overlap between blocks) categories. Each category can be collapsed if needed. All these provide fast lookup for the analyst to get to the analysis he/she is interested in.

6.2.2 Data Storage and Analysis

Here we describe our data storage and analysis module of Epiviz Feed.

6.2.2.1 Data Storage and Operations

The backend architecture stores the genomic dataset into a relational database using a two-table scheme. Each dataset is stored as two tables, one stores the data attribute values for each genomic region and the other table contains annotations for each sample in the dataset. While this structure of relational database may not be optimal in all query cases, it is suitable and effective for statistical analysis between two measurements. We implemented a service layer (epiviz data provider) that queries the database and processes requests from the application. The computational server establishes a persistent web socket connection for managing interactions between the feed and results from various statistical methods. Because of the flexibility and modularization in the data provider and the computational server, new data sources can be added easily when needed, and new statistical methods can be added or changed without affecting the entire system. This makes stages in genomic analysis pipeline independent and easily interchangeable. The analyst can either deploy the database server and web-based framework on their local machine, or use the data from the Epiviz data provider hosted at UMD (`http://epiviz.cbcb.umd.edu`).

6.2.2.2 Data Analysis Module

The statistical analysis of genomic data usually requires a significant amount of time and computing resources. Our proposed approach offloads all these operations and processing onto the server-side, which alleviates the high workload on the web browser and improves visualization performance. This motivates the design of proactive analysis to be adaptive w.r.t. target region of data. Specifically, the server starts running the computation as soon as the region is specified. Once data region of interest changes, the server restarts the computation for new region immediately.

Each type of analysis is modularized and independent. When one module fails running, it will not affect other analyses. One type of analysis may yield multiple results, which aims for every possible combination of data features or attributes. For example, Pearson correlation module would be applicable to each combination of gene expression or DNA methylation with different tissue types, as well as a mixture of the two. The proactive analysis mainly reveals the underlying relationship between measurements within the genomic data, which also provides various chart types: e.g., standard blocks track charts and scatterplots for blocks and gene expression data from Epiviz; but also scatterplots between blocks and generated methylation data. The details are in "Methods" section.

6.3 Datasets

We utilize human transcriptome data from the Gene Expression Barcode Project [140], which contains samples from 105 different tumor and normal tissues. We also incorporate methylation signal data generated by Timp et al. [141] from 6 different tissue types, including both normal and tumor samples. We selected four tissue types for our proactive analysis: colon, thyroid, breast, and lung, across two different conditions: tumor and normal. Overall, the target dataset contains over $50,000$ rows of expression data per gene and over $480,000$ rows of methylation data at specific locations in the human genome. We also include regions of differential DNA methylation between tumor and corresponding normal tissue (referred to as "blocks" in the literature). The number of blocks ranges from $1,000$ to $2,000$ regions across different cancer types.

Methods	**Data**	**Chart**	**Sample Description**
overlap	methylation blocks	bar chart	The overlap between colon blocks and lung blocks is 0.17 ($pvalue = 0.04$).
Correlation	methylation diff.	scatterplot	The correlation between Collapsed Methylation Diff breast and Collapsed Methylation Diff lung is $0.49 (pvalue = 0)$.
Correlation	expression	scatterplot	The correlation between Expression $breast_tumor$ and Expression $lung_tumor$ is $0.96 (pvalue = 0)$.
T test	exp. & methy. block	boxplot	The $t-test$ between Expression $thyroid_normal$ and thyroid blocks is 1.58 ($pvalue = 0.14$).
Correlation	exp. & methy.	scatterplot	The correlation between Expression $breast_normal$ and Probe level Meth $thyroid_normal$ is -0.07 ($pvalue = 0.61$).
Correlation	exp. diff. & methy. diff.	scatterplot	The correlation between Expression $colon_normal$ and Collapsed Methylation Diff colon is 0.17 ($pvalue = 0.23$).

Table 6.1: Example computational modules with corresponding data and chart types. We have currently used algorithms working with one or two data attributes in our computation engine. Brief textual descriptions for each module are also listed.

6.4 Methods

In this section we describe in detail the computational algorithms, and how to achieve the quantitative measures of statistical significance of the hypothesis test results in the feed. The focus of our analysis is on three raw data modalities: gene expression, DNA methylation at specific genomic location, and blocks of differential DNA methylation between tumor and corresponding normal tissue. We are interested in understanding mechanisms in which DNA methylation regulates the expression of genes of interest in cancer and corresponding normal tissue, and whether these mechanisms are consistent across different tumor types. To understand these mechanisms, statistical inferences are used based on measuring correlation between DNA methylation and gene expression within specific tissues (understanding mechanism within tissue), correlation between expression or DNA methylation across tissues (understanding the consistency of mechanism across tissues), and overlap of regions of differential methylation in cancer (understanding the consistency of mechanism across tissues). The details of these methods are stated below.

6.4.1 Promoter DNA Methylation-Gene Expression Correlation

DNA methylation is the best understood epigenetic mechanism of gene regulation. Measuring the correlation between DNA methylation and expression in a specific tissue (normal and tumor) provides insight into this mechanism for specific genes in a tissue of interest.

6.4.2 Methylation Block Overlaps

Identifying genomic regions where DNA methylation is statistically different between tumor and corresponding normal tissue is essential to understand the role of DNA methylation in cancer. Once these regions of interest are identified for each tumor type of interest, computing the overlap of these regions across tissues provides insight about the consistency, or uniqueness, of this mechanism across cancer types, which is a characteristic of biological importance. We compute block overlap between pairs of tissues (based on proportion of genomic extent in which the blocks overlap relative to the proportion they do not), for a specific genomic region. It is worth noting that Fisher's exact test [142] has been applied to obtain statistical significance of the overlap results, which will be used in the ordering of the results shown in the feed.

6.4.3 Gene Expression and DNA Methylation Correlation

Correlation between gene expressions, or DNA methylation between tissue pairs within a genomic region indicates similarity in gene regulation between tissue types. Similarities of interest are those tumor or normal types that show similar gene regulation.

To ensure the generalization of proactive analysis on genomic data, Pearson correlation, and corresponding significance test is applied to the following types:

1. Gene expressions from normal and tumor tissues of the same type, which tests whether gene expression has an effect on the tissues to be normal/tumor;

2. Gene expressions from normal/tumor of different tissue types, which identifies whether different tissues have a similar gene expression;

3. DNA methylation of different tissue types, which shows whether methylation is the same across tissues;

4. Methylation and gene expression of the same or different tissues, which tests what's the effect that gene expression has w.r.t. methylation;

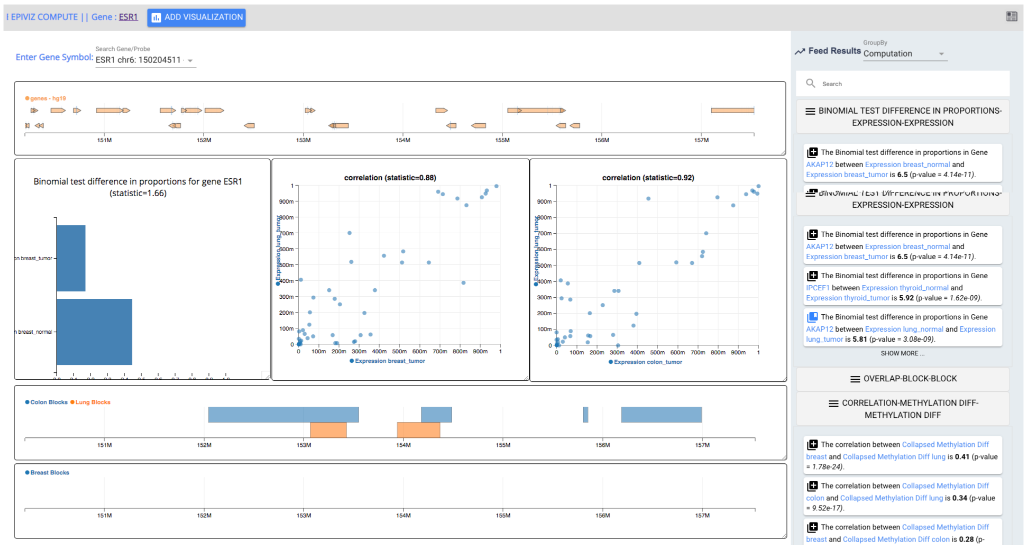
5. Methylation difference and gene expression difference of the same tissue;

6. Binomial test difference in proportions per gene within the region.

Correlations between various tissues and conditions on gene expression datasets are easy to compute since the data is at the gene level. To compute correlation between methylation and gene expression, we align base-pair (bq) methylation data with expression data. For every gene in the expression data, we extend the genomic region for the gene by 3000bp downstream and 1000bp upstream and calculate the average methylation value. This would allow us to compute correlations between these two different types of measurements.

6.4.4 Statistical T-test for Differential Expression or Differential Methylation

Similar to correlations, statistical t-test is often used to determine if two sets of data are significantly different from each other. This is important when multiple hypothesis has been made and the analyst would like to evaluate whether the



BINOMIAL TEST DIFFERENCE IN PROPORTIONS-EXPRESSION-EXPRESSION

The Binomial test difference in proportions in Gene AKAP12 between Expression breast_normal and Expression breast_tumor is **6.5** (p-value = *4.14e-11*).

BINOMIAL TEST DIFFERENCE IN PROPORTIONS-EXPRESSION-EXPRESSION

The Binomial test difference in proportions in Gene AKAP12 between Expression breast_normal and Expression breast_tumor is **6.5** (p-value = *4.14e-11*).

The Binomial test difference in proportions in Gene IPCEF1 between Expression thyroid_normal and Expression thyroid_tumor is **5.92** (p-value = *1.62e-09*).

The Binomial test difference in proportions in Gene AKAP12 between Expression lung_normal and Expression lung_tumor is **5.81** (p-value = *3.08e-09*).

SHOW MORE ...

OVERLAP-BLOCK-BLOCK

CORRELATION-METHYLATION DIFF-METHYLATION DIFF

The correlation between Collapsed Methylation Diff breast and Collapsed Methylation Diff lung is **0.41** (p-value = *1.78e-24*).

The correlation between Collapsed Methylation Diff colon and Collapsed Methylation Diff lung is **0.34** (p-value = *9.52e-17*).

The correlation between Collapsed Methylation Diff breast and Collapsed Methylation Diff colon is **0.28** (p-

Figure 6.4: A screenshot of example use case when the analyst uses the tool to analyse genomic data within gene ESR1.

hypothesis is significant. We compute a t-statistic for expression or DNA methylation between pairs of tissue within a genomic region. Besides, since the blocks data is derived from methylation (indication of whether methylation value is high or not), the statistical significance between blocks and gene expression within the same tissue is another type of analysis in the framework that is of interest to the analyst. We do this to measure the dis-similarity in gene regulation across pairs of tissues. As above, dis-similarities of interest are those tumor types that show different gene regulation, as well as normal tissues that exhibit different gene regulation. $p - value$ obtained from the test is also used as an ordering rule in the feed.

6.5 Use Case: Interactive Analysis of ESR1 Regulation across Tumor Types using Proactive Computations

Expression of estrogen receptor, for instance of Estrogen receptor 1 (ESR1) is frequently observed in breast cancer and is an important predictor of efficacy for certain therapeutic agents [143–145]. Here we present a use case where a data analyst may use our example cross-tumor in our instance of Epiviz-Feed to understand how ESR1 and other genes near the ESR1 locus are epigenetically regulated across different tumor types. The analysis workflow we discuss corresponds to the EpivizFeed workspace shown in Figure 6.4.

The data analyst would first navigate to the ESR1 locus by typing the gene name in the search box, trigering proactive cross-tumor integrative analysis for genes within the 30 consecutive genes around the ESR1 locus. The feed on the right is populated as computations are finished. The analyst first observes a number of differentially expressed genes (e.g., AKAP12 [146]) between breast tumor and normal tissue. They observe that difference in ESR1 expression is not on the top list of results listed but they can use the feed search bar to find ESR1 results and add a scatter plot visualization to represent it.

They next follow up on cross-tumor results observed in the feed. First, they observe high correlation in tumor-normal methylation differences in breast and lung, and high correlation of tumor expression in breast and lung suggesting. By adding associated visualizations for these two tests they can hypothesize potential similar gene regulation in both tumor types [147]. Similarily, they explore similar results between colon cancer and lung cancer, which is also the overlap of hypo-methylation blocks between colon cancer and lung cancer, which is displayed as a blocks track in the workspace. To compare with breast methylation in the same region, they manually add tracks for breast hypo-methylation block data and differential methylation signal for breast, colon, and lung. Based on this type of workflow, that integrates proactive computation with exploratory visualization, analysts can explore a variety of statistical results along with the underlying data that may support these results to discover patterns from integrative data analysis.

6.6 Discussion

Epiviz Feed couples automatic analyses with visual exploration, reducing the time cost for the analyst to manually run the tests. It integrates confirmatory and exploratory data analysis into one single tool. Besides the functionality of interative genome browser that is already offered in Epiviz, the analyst can transit between the two types of analyses easily using either visualizations in the main window or results from the feed. We regard the tool as a first step towards a more intelligent genomic data analysis tool. One thing to note is that as multiple tests are running on the same datasets, the chance of false positives (also referred to as “p hacking”) increases. We attempted to mitigate this through making the p-values available in the results. It is important to note that our proposed tools are intended for exploratory data analysis and that any significant hypotheses should be followed with a formal, controlled study to confirm or deny. The other possible way would be using adaptive analysis in differential privacy to consider associated pitfalls.

While Epiviz Feed currently supports analysis and visualization within selected genomic region by the analyst, another interaction would be to use gene set as focused region rather than genomic locations, which will be useful when analyst is working on comparative study of the genomic data across differen genes. The other continued research work is to support more advanced statistical and machine learning algorithms, such as ELMER [148], to help analyst find insights. This will provide in-depth analysis and patterns within the dataset. Another idea is to find patterns that is similar to the charts that the analysts are interested in across all the gene sequences. This will provide broader analysis and faster exploration across genomic data. We also envision a collaborative version of Epiviz Feed essentially serving as an "Analyst's Facebook" in that it would allow a team of analysts to work together and share their findings using a connected feed.

6.7 Conclusion

In this chapter, we presented the design and structure of applying proactive visual and statistical analysis framework in Epiviz, a web based interactive visual analysis tool for genomic sequence data. We gave an overview of the motivations and design rationales for providing automatic analytical results to the analysts, thus bridging the gap between the analyst and computer. It alleviates the analyst from creating visualizations manually through presenting the results using "Feed" on the right of the Epiviz visualization tool, which is inspired from social media post. The feed will update dynamically when new analysis comes in. The analysis results and tests are provided in such a way that is easier to understand. The analyst can filter results based on their needs. We use the proactive tool to analyze existing dataset and highlight how the feed helps in the visual and statistical data exploration. One contribution is to utilize computational power to reduce the time spent on creating visualization, such that analyst can focus on analyzing the data and find out insights.

Chapter 7: Leveraging User Interaction in Visual Analytics for Computational Steering

As per the definition of visual analytics (VA) [8], many VA applications require significant computations—such as clustering [9], word embedding [10], and inferential statistics—to be run on new datasets prior to presentation to the user. However, real-world datasets are increasingly reaching a volume and complexity where such computation can be forbiddingly costly in terms of processing and time, resources which the analyst may not be able to spare. To remedy this, big data analytics [87] is increasingly turning to partial, progressive, and incremental methods, where instead of waiting for computation to finish prior to viewing the data, the user is shown an intermediate view of the data that is continuously updated throughout the computation [88, 89]. While advanced database techniques can provide reliable partial results of even large datasets [33], we could be using our computational resources more efficiently if we knew which part of the data the user was interested in. For example, given ten years of fine-grained stock market data, clustering stock trends for each time segment starting from the beginning of the recorded time period is inefficient if the user is only interested in the stock market from last year. Unfortunately, most current interfaces for this kind of *computational steering* [75, 149] of time-consuming computational processes often require significant expertise of the computation itself, which only few data analysts possess.

In this chapter, we propose Sherpa, a method that can leverage user's attention to implicitly derive priorities for computational operations of the dataset. Sherpa provides a *data space view* (Figure 7.1) where the user can control their current locus of attention using a *navigation window*. For example, in the 10-year stock market data, the user may pan and zoom their navigation window to focus solely on the last year trend in the dataset. It uses the dynamically changing navigation window in the data space view to implicitly derive the priority of computation for each portion of the dataset. The Sherpa scheduler will prioritize finishing the background calculation for those areas of the dataset that the user has expressed an interest in using the navigation window. The main visualization, which is specific to the particular application, will show a progressively updating view of the currently selected subset of the data as computation proceeds. Priorities decay over time, allowing the user to change their focus throughout the analysis process.

The Sherpa method is independent of applications, and could be applied to any dataset provided the computations can be localized to specific regions of the data, such as the stock market, time-series data, and local network metrics. We have implemented Sherpa for a human genomics application, where multiple data modalities—gene expression and DNA methylation—across four cell types (colon, lung, breast and thyroid) and their corresponding normal tissues are spatially indexed over genomic position across 23 chromosomes (over 3B possible positions in total). The computational processes of interest in this application are statistical in-

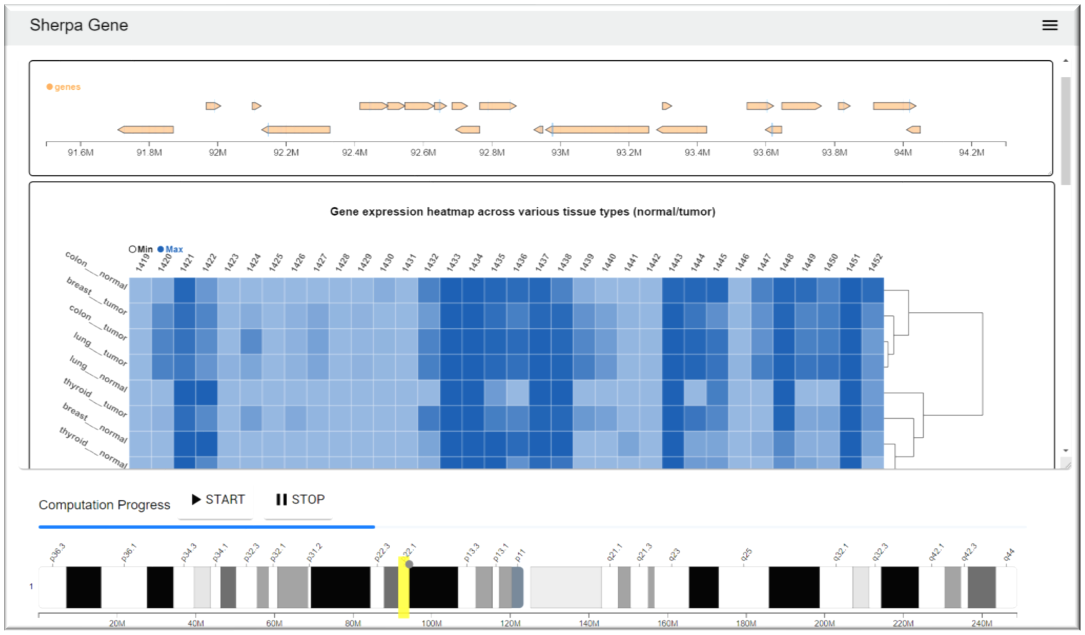

Sherpa Gene
genes
Gene expression heatmap across various tissue types (normal/tumor)
Min
Max
Computation Progress
START
STOP

Figure 7.1: Sherpa Gene: a web-based visual analytics application for genomics incorporating attention-based computational steering. The gene track and gene expression heatmap display the user's current focus. The ideogram (bottom) serves as the data space view on which the user controls the yellow navigation window, which governs computational priority.

ferences that reveal mechanisms underlying gene regulation (in particular, the role of DNA methylation in regulating gene expression), how those mechanisms change in tumor relative to the normal tissue, and how they are manifested across four cancer types under study. The progressively updated visualization (Figure 7.1) shows a track displaying gene location and structure within the focused genomic region, another track displaying genomic blocks of significant methylation difference between tumor and normal tissue, and a heatmap of gene expression across multiple tissues, with multiple small scatterplots and block tracks gradually being added to the main visualization space for statistically significant tests (based on correlation and block overlap computations). The data space view uses the spatial position within the chromosome to order data, and the user's movement of the navigation window will change the priorities of which part of the dataset to run computations on.

We have evaluated our Sherpa implementation for the genomics application under three conditions: (1) a classic static condition, where only the final computation is shown to the user (which serves as a baseline); (2) a progressively updating condition, where the display updates as computation proceeds but where the user cannot steer the computation; and (3) a Sherpa condition, where the user's navigational behavior on the sequence will steer the order of computation. In our study, participants were asked to answer high level questions about specific aspects of the data. Not surprisingly, our results show that implicit computational steering using the Sherpa approach provides significant time improvements for tasks that are specific to known gene locations (e.g., specific genes of interest). This suggests that computational steering can be beneficial for visual analytics, even when the user lacks the expertise to explicitly control the computation.

7.1 Attention for Computational Steering

The Sherpa[1] model is an implicit form of computational steering for priority-based processing of a dataset based on user's attention. The intuition behind the model is to prioritize computations on those areas of the dataset that the user deems important. We derive user attention from the location and dimensions of an

[1]The Sherpa people are native to Tibet and are known for their elite mountaineering skills. They were instrumental to early exploration of the Himalayas; hence, our use of the word to signify a "guide" for data exploration.

interactive *navigation window* on an overview representation of the dataset (*data space view*). Figure 7.2 gives an overview.

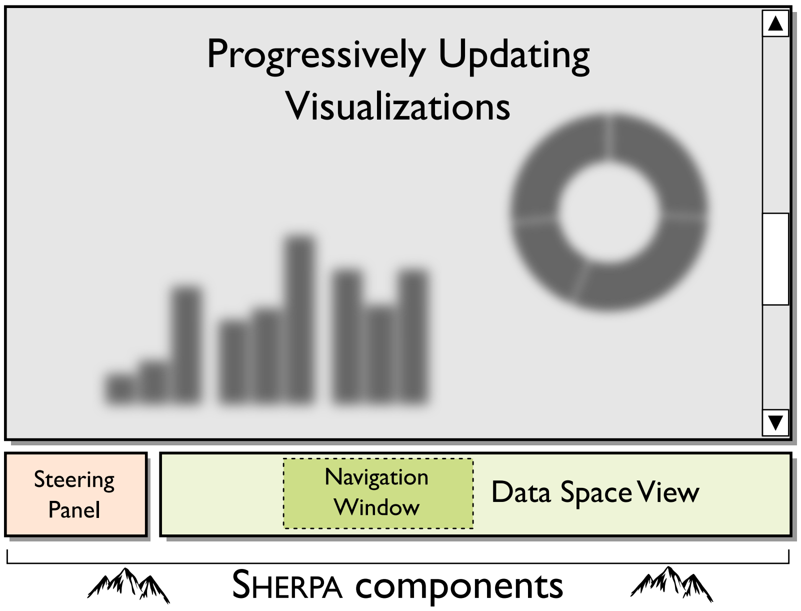


Figure 7.2: Overview of the Sherpa user interface components.

7.1.1 Basic Model

Sherpa is a general model that can be instantiated for specific applications, datasets, and computations. It makes a certain number of assumptions about the application:

- A *dataset* where position has meaning;
- A parallelizable *computation*; and
- A progressively updating *visualization*.

First of all, Sherpa requires a *dataset* with natural location-based semantics; this could either be truly spatial, such as for locations on a map or positions in a gene sequence, or temporal, such as positions in time. Of course, the dataset should be of sufficient size where it cannot just be trivially processed prior to shown to the user; in such case, the Sherpa method (or any other PVA method) is not necessary.

Second, Sherpa requires a corresponding *computation* on the dataset that can be performed on data items in random order. In other words, the computation must be *parallelizable* so that computational results for a specific subset has no dependencies to results for other parts of the data. Basically, Sherpa has the same limitations as the ProgressiVis toolkit [90], which discusses different classes of algorithms that are suitable for progressive implementations. Note that it is possible that computations could be restricted to chunks of items instead of individual item, as long as there is a large enough number of chunks so that their computational order is important.

Third, the method requires a *visualization* of the dataset that (1) can represent a specific subset of the dataset, and (2) can be progressively updated over time as new calculations are completed. The former property is required so that the user's navigation in data space actually affects the main visualization view (otherwise there is no purpose of the user to navigate in the data space view); the latter is also needed so that the view can be refreshed as new results are produced.

7.1.2 Steering Functionality

Given an application that fulfills all of the above assumptions, Sherpa maintains a central priority queue for each data item (or chunk of items). The priority queue is initialized so that each item has the same priority, and the items are ordered based on the semantic position in the dataset. Thus, if the priorities are never changed, the computation will proceed from whatever is defined as the "beginning" of the dataset to its "end" (this varies between applications; for a stock market dataset, the beginning is the time the data commences and the end is where it stops; for genomics, the beginning is the start of the gene sequence, and the end is where it stops).

Starting from when the Sherpa application is launched, a concurrent computational engine will launch and run the computation based on the priority queue. A practical Sherpa implementation will implement this engine either as a background, multi-threaded process, or on the server side.

The Sherpa user interface includes a basic *steering panel*, modeled after work by Badam et al. [88], which provides simple computational steering operations that interface with the computational engine: starting, stopping, and pausing the computation. The panel also shows the current progress.

7.1.3 Data Space View

Given a dataset with location semantics, the *data space view* is a spatial representation of the dataset. Depending on the application, data space view can be 1D or 2D in nature: for a gene sequence or timeline, for example, it would be represented by a single dimension, whereas for a geographic map or spatial data structure, it would be two-dimensional. A key aspect of the data space view is that it communicates the position in the dataset using labels, ticks, or grid lines (or a combination of these), allowing users to orient themselves and navigate accurately in the spatial dimension.

In addition to displaying the extents of the dataset, the data space view also conveys the following components:

- A *summary visualization* of the underlying data, such as an average stock market index, gene sequence delimiters, or geographic summary statistics;
- A *priority curve* displaying the relative computational priority for each segment of the dataset, indicating which segment will be computed next; and
- A *progress indicator* that gradually fills in as computation for each segment of the dataset is completed.

7.1.4 Navigation Window

Finally, the *navigation window* in the data space view represents the user's attention on the dataset, which will guide the computational steering. It is represented by its *extents*: for one-dimensional data space, this is a simple interval (e_{min}, e_{max}), whereas for a two-dimensional one, it is a bounding box $(x_{min}, y_{min}, x_{max}, y_{max}$. As such, the navigation window is initialized at the beginning of the exploration to cover the entire dataset $(0, 1)$ or $(0, 0, 1, 1)$ (inclusive).

Actually, interacting with the navigation window can be done by *panning* (which means translating the extents) or *zooming* (changing the size of the window $e_{max} - e_{min}$). The main visualization window should be synchronized to always display only the portion of the data that is currently selected by the navigation window. Typically this can be done in several ways: (1) move the window by dragging on the window itself using a mouse or finger touch (panning); (2) change window dimensions by dragging on one of the window borders using a mouse or finger (zooming); (3) change the window size by rotating the mouse wheel or pinching (zooming). All the while, the extents should be kept within the range $|0, 1|$.

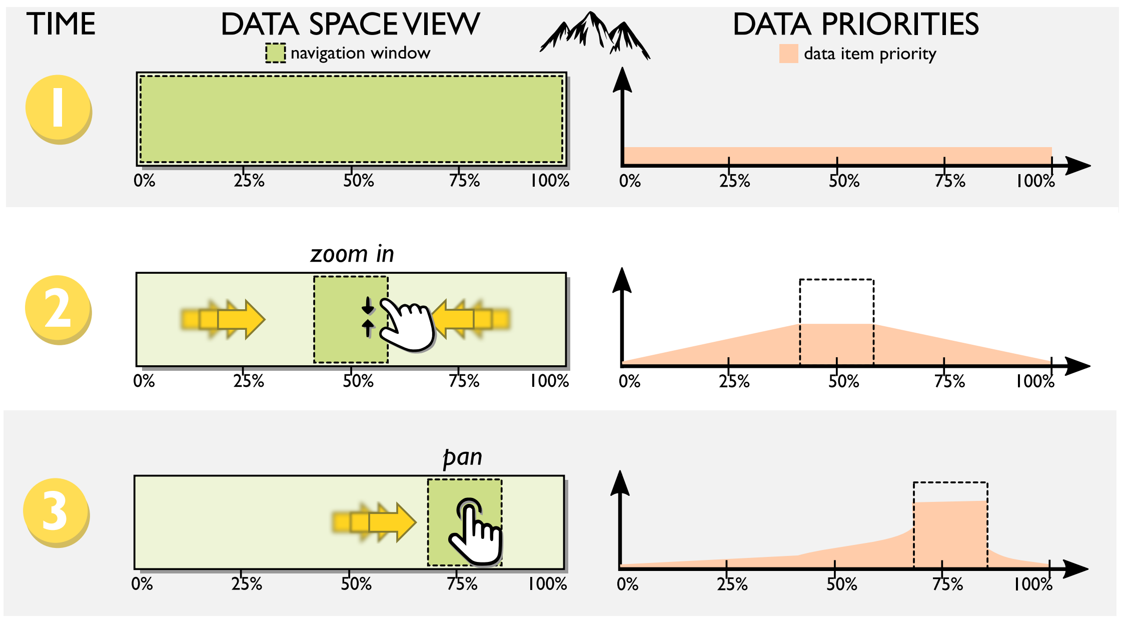


Figure 7.3: Mining attention as navigation behavior over time.

7.1.5 Mining Attention

The final piece of the Sherpa method is leveraging the user's attention. We use the navigational behavior of the user as a proxy for their attention. We base this on the intuition that the user's interaction with the navigation window in the data space view is a representation of which part of the data the user is interested in. The behavior of the navigation window is then used to adjust the priority of each data item.

More specifically, we view attention as the position and dimension of the navigation window over time on the dataset. Let us assume that the user confers a constant 1.0 of attention on the view per time unit. If the entire dataset of N items (or segments of items) is within the navigation window (as is the initial state), then each item will be receiving $1.0/N$ of attention per time unit. No specific item will be receiving more attention than the others, leaving the priority queue unchanged. However, if the navigation window is zoomed, reducing its size to a smaller $n < N$, then all of the items still within the new navigation window will be receiving $1.0/n$ of attention per time unit. Numerically integrating this over time will enable Sherpa to essentially model user attention on the dataset (Figure 7.3).

Since attention may change over time, we also introduce a temporal decay function that gradually reduces the accumulated priority of each data item per unit of time. We have experimented with several such decay functions; perhaps the most useful approach is to use a radioactive decay function:

$$P(t) = P_0 \left(\frac{1}{2}\right)^{\frac{t}{t_{1/2}}}$$

where P_0 is the initial priority, t is the time parameter, and $t_{1/2}$ is the half-life of the priority decay. Values for specific constants will need to be determined for each application.

Finally, while we have not focused on collaborative aspects in this work, the Sherpa method does allow for modeling the attention of multiple analysts based on their navigational behavior on the data space view. This will provide a mechanism for one team of analysts to collectively steer the computation. However, the accumulation of attention over time may have to be modified to prevent one user from gaming the system by shrinking their navigation window to incur a high attention on a very small part of the dataset, thus prioritizing only their view.

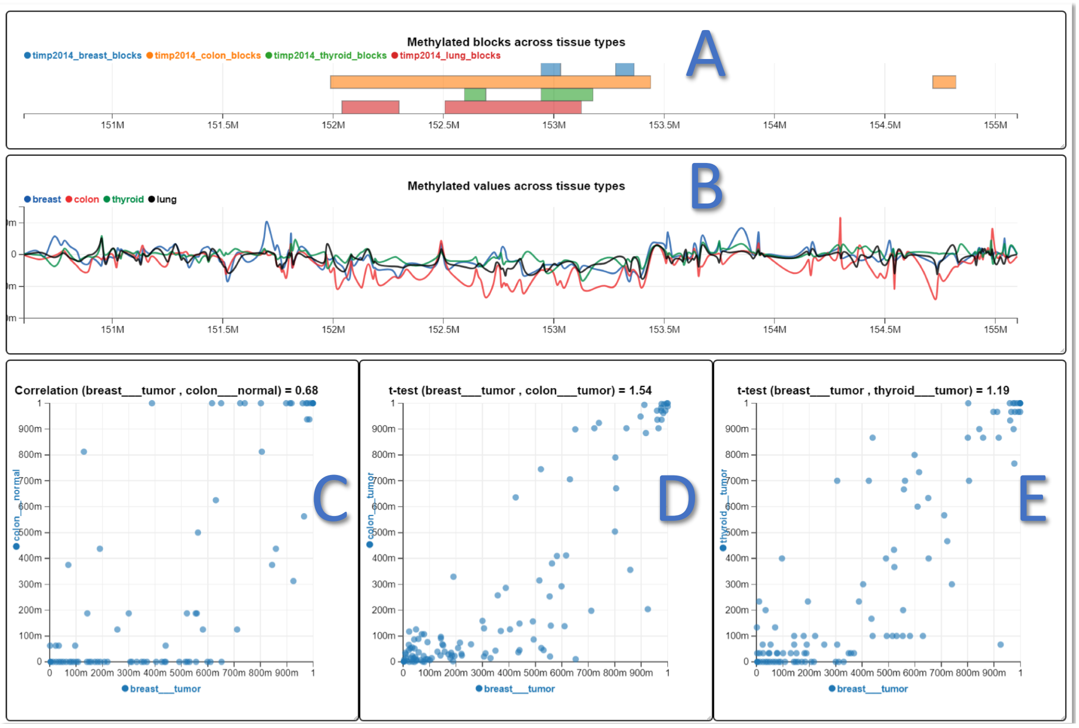


Figure 7.4: Additional example charts from the Sherpa Gene application. Genes track and heatmap for gene expression across tissue types (breast, colon, thyroid, lung) are shown in Fig 7.1. (A): Methylated Block track: indicating differentially methylated genomic regions within which DNA methylation is significantly different (according to an offline statistical inference) between tumor and the corresponding normal tissue for the four tissues under study; (B): Methylation line track: shows the difference in DNA methylation between tumor and corresponding normal tissue at specific genomic positions; (C): Scatterplot of gene expression for two different tissues, illustrating correlation between gene activity in those tissues; (D, E): Scatterplot of gene expression for two tissues, illustrating difference of gene expression in those two tissues measured by a t-statistic.

7.2 Sherpa for Genomics Data

To showcase the Sherpa framework, we developed an interactive visual analytics tool—SHERPA GENE—that uses the proposed method to support attention-based computational steering in functional genomics (Figure 7.1). This tool fulfills the general Sherpa requirements as follows:

- *Dataset:* The tool uses genomics data, gene expression and DNA methylation, which is indexed by location within the human genome.

- *Computation:* Our user group is interested in understanding mechanisms in which DNA methylation regulates the expression of genes of interest in cancer and corresponding normal tissue, and whether these mechanisms are consistent across different tumor types. To understand these mechanisms, statistical inferences are used based on measuring correlation between DNA methylation and gene expression within specific tissues (understanding mechanism within tissue), correlation between expression or DNA methylation across tissues (understanding the consistency of mechanism across tissues), and overlap of regions of differential methylation in cancer (understanding the consistency of mechanism across tissues). The computations required to calculate these statistical measures of biological importance are easily parallelizable.

- *Visualization:* We use several progressive visualizations that summarize the current focused region: a genes track indicating specific genes contained in the region of interest, a gene expression heatmap showing similarity (and dis-similarity) of expression for multiple tissues, scatterplots showing trends in expression or DNA methylation within and across multiple tissues, line tracks showing DNA methylation values at their corresponding genomic position, and region tracks showing regions of differential methylation in different tumor types from which the overlap of these regions of interest can be observed.

The interactive workflow of Sherpa Gene typically involves exploring specific genomic regions based on genes of interest. Therefore, we utilize the user's genomic location within the chromosome to steer the computation.

7.2.1 Dataset

Sherpa Gene contains human transcriptome data from the Gene Expression Barcode Project [140] for 105 different tumor and normal tissues. The database also contains methylation signal data [141] for 6 different tissue types and includes both cancer and tumor samples. We selected four tissue types in the implementation: colon, thyroid, breast, and lung, across two different conditions: tumor and normal. Overall, the database contains over 50,000 rows of gene expression data per gene and over 480,000 rows of DNA methylation data at specific locations in the human genome. We also include regions of differential DNA methylation between tumor and corresponding normal tissue (referred to as "blocks" in the literature). The number of blocks range from 1,000 to 2,000 regions across different cancer types.

7.2.2 Computational Algorithms

Our data includes three data modalities indexed by genomic location: gene expression, DNA methylation at specific genomic location, and blocks of differential DNA methylation between tumor and corresponding normal tissue. While there are many types of computations that can be applied to data of this type, Sherpa Gene implements the following:

- **Promoter DNA Methylation-Gene Expression Correlation:** Correlation between DNA methylation and gene expression of specific tissues (normal and tumor). DNA methylation is the best understood epigenetic mechanism of gene regulation. Measuring the correlation between DNA methylation and expression in a specific tissue provides insight into this mechanism for specific genes in a tissue of interest.

- **Methylation Block Overlaps:** Identifying genomic regions where DNA methylation is statistically different between tumor and corresponding normal tissue is essential to understand the role of DNA methylation in cancer. Once these regions of interest are identified for each tumor type of interest, computing the overlap of these regions across tissues provides insight about the consistency, or uniqueness, of this mechanism across cancer types, which is a characteristic of biological importance. In this application, we compute block overlap between pairs of tissue (based on proportion of genomic extent in which the blocks overlap relative to the proportion they do not), for a specific

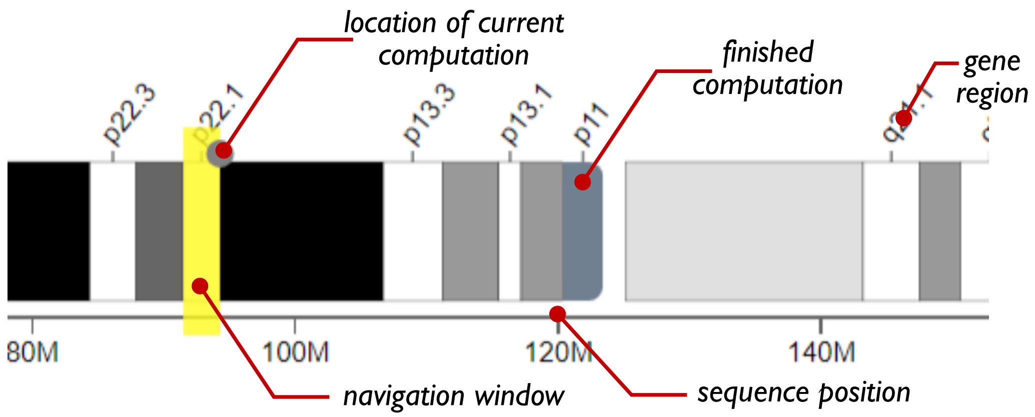


Figure 7.5: Detail of a chromosome ideogram—an idealized depiction of a chromosome—being used as a data space view in Sherpa Gene. Navigating the chromosome using this interface will steer the server-side computation as well as control the data being displayed on the browser-based client.

genomic region.

- **Gene Expression or DNA Methylation Correlation:** Correlation between gene expressions or DNA methylation between pairs tissues within a genomic region. This indicates similarity in gene regulation between tissues. Similarities of interest are those tumor or normal types that show similar gene regulation.

- **t-test for Differential Expression or Differential Methylation:** We also compute a t-statistic for expression or DNA methylation between pairs of tissue within a genomic region. This metric measures the dis-similarity in gene regulation across pairs of tissues. As above, dis-similarities of interest are those tumor types that show different gene regulation, as well as normal tissues that exhibit different gene regulation.

7.2.3 Sherpa Controls

Our prototype implements all of the Sherpa controls in a region at the bottom of the display. The steering control panel (Fig. 7.1) allows for starting and stopping the server-side computation at any time. The Sherpa data space view is implemented as an ideogram (Figure 7.5), which is an idealized graphic representation of a chromosome. The navigation window is a yellow region showing the current focus. A progress bar (a compact ideogram) shows the current status, which will gradually fill in with a transparent blue color as the computation proceeds.

Moving the navigation window on the data space ideogram will both steer the computation as well as govern which visualizations will be shown in the main view (see below). Pilot test pushes us to immediately give regions inside the navigation window in Sherpa Gene top priority. If the user does not navigate, or if the computation for a specific focus region has finished, the analyses will continue on other genomic regions based on accumulated priorities.

7.2.4 Progressive Visualization

The main view of Sherpa Gene is consumed by progressive visualizations that show the currently selected genomic region of focus (controlled using the navigation window). Instead of a single visualization, Sherpa Gene progressively add charts showing details of the computations as results are produced.

More specifically, genes track (Fig. 7.1) is shown on top of the main view by default to provide an overview of the genes within a region to the user. A heatmap (Fig. 7.1) with cell and tissue types as rows, and genes as columns allows for comparing gene expression values. Methylation block overlaps (Fig. 7.4(A)) are shown in a stacked blocks track for all tumor types. A DNA methylation line track (Fig. 7.4(B)) makes it possible to investigate changes and trends in detail. The space below these charts is used for adding scatterplots (Fig. 7.4 C, D, E), one by one, each representing correlations in expression or DNA methylation between normal and tumor tissue types, whenever significant values are found.

7.2.5 Implementation Notes

The Sherpa Gene implementation uses a server-client architecture. The client interface was developed with modern web technology: HTML5, JavaScript (JS), and CSS3, along with Polymer 2.0 [150] and the Epiviz web component library [135]. The backend server consists of: (1) MySQL database, which stores the genomic data; (2) Epiviz data provider [151], which extracts data from database; and (3) a computational server, that runs all the computations and provides a websocket endpoint using the Python Flask framework. The data provider ensures fast retrieval of the data from a MySQL server, and websocket connection enables streaming results back from server to client through chunks when one batch is finished.

7.3 User Study

The goal of the Sherpa framework is to enable an analyst to steer a computational process using their attention alone. Thus, we are building on the notion of progressive visual analytics [90], which does include both visual updates (output) as well as computational steering (input), but which does not stipulate *how* computation should be guided. Our hypothesis is that the interest-based computational steering (progressive input and output) that Sherpa embodies will perform better than just gradually updating the visualization (progressive output only). To test this hypothesis, we conducted a qualitative expert review using our Sherpa genomics implementation.

7.3.1 Participants

We recruited in total 5 participants (4 male, 1 female): 2 from a visualization group and 3 from a bioinformatics lab at our university. Participants were between 24 and 33 years of age, had normal or corrected-to-normal vision, and were experienced computer users. In particular, all participants had significant experience in visualization, bioinformatics, or computer science, and were well-versed in visualization and genomics.

7.3.2 Experimental Design

We organized each experimental session into three conditions that all expert participants were exposed to:

- *Blocked:* In this condition, the computation was completed prior to a trial commenced, thus giving the user immediate access to the full results. The participants were merely informed of the full execution time required to perform this computation (on the order of 5-6 minutes depending on genome size). The software used was our genomics prototype application, as described in the previous section, but with all progressive and steering functionality disabled.

- *Progressive output:* Here, the computation was launched at the same time as the trial started, but the Sherpa attention-based steering functionality in our tool was disabled. Thus, the computation proceeded from the beginning of the genome until it reached its end (which, as stated above, took approximately 5-6 minutes). During this time, the main visualizations in the genomic application were progressively updated, and participants could interact with the tool to perform tasks. Participants could use the data space view and navigation window to move around the dataset, but their navigation behavior was not used to modify priorities.

- *Progressive output & input:* Finally, in this condition, we enabled the full Sherpa functionality, including attention-based computational steering. Computation started simultaneously with the trial, and participants had full access to all features of the tool.

7.3.3 Task and Procedure

For the purposes of the expert review, we asked our participants to answer a collection of five tasks related to a specific chromosome. With three conditions, we created three separate such sets of tasks for three different chromosomes. These tasks were balanced across conditions, but the order of conditions was always the same to enable pre-computation to finish before each session starts. Given that our study is qualitative, we think that the lack of counterbalancing had little impact on our results. In fact, presenting the non-progressive version first, where all data is immediately available, provides a useful baseline comparison for our participants.

Participants were encouraged to solve tasks in any order. We encouraged participants to follow a think-aloud protocol, and recorded their utterances. The experimenter took extensive notes, and the prototype software stored an interaction log. Each session lasted between 45 and 55 minutes.

The tasks were exclusively location-based in nature, i.e., about genomic regions containing a specific gene of interest that a participant could navigate using the data view. While this certainly favors the Sherpa method, where navigational behavior affects computation order, this was precisely the purpose of our expert review. We wanted to understand the utility of this method rather than study completion time and task accuracy for a fully ecologically valid use case. We leave such studies for future work.

7.4 Results

We first report the objective results from the evaluation as well as the think-aloud comments, then explain the outcome of observations and post-study interviews.

7.4.1 Performance Results

All five participants successfully finished the tasks in all three experimental conditions. When first starting the application, participants all experimented with the data space view and navigation window to understand the steering functionality. They were pleased to see results gradually update as computation proceeded in the background. One participant said, *"I don't care about how the computation works, but I think showing intermediate results is absolutely necessary."*

Compared with the progressive output condition, the Sherpa functionality gave participants more perceived control over the visualizations, thus making it easier to access the results. While we did not measure the exact time for individual tasks, we observed that participants spent less time in total to finish the tasks under Sherpa condition than with progressive output. 3 participants said they were frustrated when they realized there was no interactive steering in the situation with only progressive output. With respect to the blocked condition, where computations were precompleted, the overall usage time for finishing all five tasks was only slightly faster than the Sherpa condition.

7.4.2 Usability Feedback

Overall, participants were all very interested in the Sherpa framework and praised our effort combining computational steering and visual analytics. All participants thought Sherpa Gene is very useful in exploratory analysis, and 2 participants said that it's even more helpful for search tasks, in which one needs to explore multiple regions within the data, such as *"find the region that has the highest correlation between colon tumor and normal tissue."* One participant mentioned that the prototype application *"makes me motivated to control the computation,"* essentially forming an analytical partnership between the user and the computer [123]. When given a task where participants needed to look into multiple regions to find the answer, e.g., a search task, they would navigate to those regions and get familiar with the results, which would be progressively computed based on the navigation. In other words, the use of attention as a prompt conformed well with our participant's intuition when foraging for information [152]. Furthermore, one participant said, *"different orders of exploration [computations] may produce more insights, which users can control easily [in Sherpa]."* From our observations, we also saw that participants often selected diverse regions seemingly at random (many not related to the task) to merely gain understanding about the data.

With respect to the conditions in the study, all five participants thought Sherpa condition was superior to the other ones. Three stated that the blocked condition with preloaded computations was not appropriate because a common task is to just get a quick view of a small region in the dataset, and they would not want to wait for all computation to finish. One noted that preloading all computations in one shot is not feasible. Precomputing results may also incur unnecessary waiting time since different tasks may need different computational modules. Surprisingly, the progressive output condition was the least preferred among all three conditions. One participant claimed that he would not want to use a tool with no steering: *"when I select a region, I'd like to see the results [in that region], that's the purpose of my* *selection."*

One participant also suggested the tool would be very useful to understand gene regulation in disease settings, where analysts would navigate to genes with related function but may not be located in the same genomic region.

7.4.3 Points of Improvements

Participants also provided valuable suggestions on how our tool can be improved. Two participants suggested that it would be useful to maintain a history of user-selected regions. This may be particularly helpful for complex tasks that require comparing data across multiple regions. In a way, our numerical integration of priority over time does serve this purpose, as it will "remember" parts of the data space the user has visited, and prioritize their computation.

One participant also raised a concern about the trade-offs between how much computational power the user wants to leverage and how fast steering should work. While this is an interesting question, it is beyond the scope of this project.

7.5 Discussion

Here we attempt to explain our results for implicit computational steering and then discuss limitations of our work.

7.5.1 Explaining the Results

The Sherpa framework provided a significant advantage for participants solving location-based tasks, particularly when the task involves searching through multiple regions. Compared with only progressive visualization and precomputed conditions, participants were more engaged in the exploratory data analysis process in the Sherpa condition. This is not surprising: steering, even implicitly using navigation behavior, provides direct control over the computation. With no steering, participants could not see the outputs for a region until the computation for that region was finished.

However, we were surprised to see only a small difference between Sherpa and the precomputed condition, where all results were immediately available. One explanation is that in Sherpa Gene, the individual computations are lightweight and can be finished quickly, which means that navigating to a specific region will quickly yield results. Initial results would come in within just a few seconds, which would not be much slower than for the precomputed condition. A more time-consuming server-side computation would not be able to yield the same near-instant responsiveness.

Another surprising observation is that all participants preferred the blocked over the progressive output condition. This indicates that interaction is an essential part of progressive visual analytics (PVA). In fact, our results cause us to speculate that progressive visualization without steering may actually have a negative effect on user experience.

7.5.2 Limitations

As mentioned before, the tasks in our evaluation were all location-based questions. This was intentional to elicit findings specifically about Sherpa's specific steering functionality, but means that the study is not fully representative of realistic functional genomic analytical workflows. Future studies should include more general and ecologically valid tasks.

While we are using a real-world genomics dataset [140], our computations were only simplistic. We select them to be parallelizable so that they would fit within the Sherpa framework, which is certainly not true of all algorithms used for functional genomics. Nevertheless, we believe they were complex enough to generate realistic exploratory tasks that enabled observing the usage of our Sherpa Gene application. Besides, the visualization components in Sherpa Gene draw from the Epiviz framework [135], which has been proven to be easily scalable and reusable to other genomics datasets.

Finally, utilizing navigational behavior for computational steering is susceptible to a variant of the "Midas touch" problem in HCI [153]: distinguishing between interaction for exploration (implicit) vs. interaction for selection (explicit). Put differently, some navigation behavior may not be a direct indication of interest, but rather merely a form of *epistemic* action [154] (as opposed to *pragmatic* ones) that helps the analyst understand the scope and shape of the dataset. In fact, we saw indications of this in our study: some participants would idly "click around" on the ideogram bar to view various regions. We see this as a caution against attempting to infer too much from navigation behavior alone.

7.6 Conclusion

While the emergent research topic of progressive visual analytics (PVA) provides an exciting, realistic, and future-proof method for managing even massive datasets in an interactive workflow [89, 90], existing PVA systems have—with a few exceptions [10, 88]—largely left the input side of the equation unexplored. To remedy this, we have proposed the SHERPA method for computational steering in PVA based on user navigation, thus eliminating the need for the analyst to explicitly control the computation order. We implemented a genomic application, Sherpa Gene. Results from our expert review with bioinformaticians using Sherpa Gene for genomics analysis provide empirical validation for our approach; while obviously having immediate access to computational results is preferable, our participants felt that the Sherpa model was more empowering and efficient than merely seeing progressive visual updates.

Chapter 8: Conclusion and Future Work

This thesis presents approaches of improving data analytical procedure on how to understand large scale dataset quickly and effectively. Specifically, we aim to enhance the efficiency of interactions and sensemaking in visual analysis, through faster rendering and processing techniques, as well as user guided automatic computations. We tackle the problem from both sides of data science: the computer and the analyst.

To push the computer to process and present the data quickly and automatically, we first evaluated SVG based visualization rendering performance on the browser in Chapter 3, to understand the scalability of SVG visualizations. We studied rendering time for scatterplots and parallel coordinate plots in three distinct phases of SVG rendering process: (a) DOM manipulation, (b) style and layout computation, (c) and pixel rasterization (painting). We documented findings from the evaluation to achieve better performance when rendering SVG visualizations, including: (1) as anticipated, faster rendering can be achieved by lowering the number of elements; (2) faster DOM manipulation can be achieved by reducing coordinate precision; (3) faster styling, and pixel rasterization can be achieved through specific CSS style properties: by optimizing stroke-width, opacity, radius, etc. We have also identified a set of best practices for guiding visualization developers to make informed decisions on how to achieve faster SVG rendering.

In Chapter 4, we proposed a framework that leverages local devices for visualization processing and computations. We also introduced VisHive, an instantiating JavaScript toolkit for constructing web-based visualization applications that can transparently connect multiple devices—called cells—into such ad-hoc clusters—called a hive—for local computation. Hives are formed either using a matchmaking service or through manual configuration. VisHive is built entirely using current web technologies, runs in the native browser of each cell, and requires no specific software to be downloaded on the involved devices.

In order to better partner the computer with the analyst for mixed-initiative analysis, in Chapter 5 and Chapter 6, we present DataSite, a proactive visual analytics system where the burden of selecting and executing appropriate computations is shared by an automatic server-side computation engine. Salient features identified by these automatic background processes are surfaced as notifications in a feed timeline. We validate the system with a user study comparing it to a recent visualization recommendation system, yielding significant improvement, particularly for complex analyses that existing analytics systems do not support well. We have further integrate the DataSite system to a genomic data analysis tool to formalize Epiviz-Feed. With the help of this, the analyst could reduce the time spent on creating visualization, such that he/she can focus on analyzing the data and find out insights.

Finally, in Chapter 7, we present Sherpa, a computational steering system managed by user interaction for progressive visual analytics that automatically prioritizes computations performed on the dataset based on the analyst's navigational behavior. Our example web-based client/server implementation of Sherpa provides computational modules for genomic data analysis, including correlation, t-test statistics, as well as similarities measures related to gene expression and methylation data. The position and dimension of the navigation window on the genomic sequence over time is used to prioritize these computations to genomic regions favored by the user. In a study with experienced genomic and visualization analysts, we found that Sherpa provided comparable accuracy to the situation where all computations were completed prior to analysis, while enabling shorter completion times.

8.1 Future Work

While in this dissertation, we have fulfilled some of the requirements for efficient presentation and interaction of VDA, there are still a lot of space to improve the research in this area. Here is the summary of potential future work that extends beyond this dissertation.

8.1.1 Performance Evaluation on Web based Visualization

As introduced in Chapter 3, we have achieved thorough knowledge of the browser rendering workflow when creating SVG visualizations, as well as devised ways of collecting detailed measurement data. We hope to expand this knowledge into a publicly available web service, which users can visit using their own device and choose a visualization technique in order compute the DOM manipulation factor, styling factor, and painting factor for specific use cases similar to our measurements. Such a web service would be of particular usefulness for visualization developers to get a first-hand experience of how specific devices and browsers perform when rendering SVG visualizations.

In general, further work would address the other side of web performance evaluation: for example, the use of HTML5 Canvas and WebGL. Many third-party APIs and toolkits for high performance rendering in JavaScript use Canvas or WebGL (which is implemented on top of Canvas) to achieve much faster rendering speeds, but at the cost of loss of high level scene graph properties of SVG. It would be interesting to dive deeper into these tradeoffs and what they mean for web based visualization in general.

8.1.2 Visualization Computations on Multiple Devices

With respect to utilize available computational resources, we have also seen many potential refinements and improvements of the VisHive toolkit presented in Chapter 4. For example, one possible extension is to make all steps of the visualization pipeline in a distributed manner, including not just data transformation and the visual encoding, but also the view transformation and input management. The other idea is to investigate how to use the slave cells not just as headless computational units, but also for collaboration (for multiple users) or for supporting the main device with additional views and input surfaces (for a single user with multiple devices). Finally, we would like to study the usability aspects of firing up multiple devices to offload a main device, and how this discovery and handshaking process can be streamlined.

8.1.3 Proactive and Mixed-Initiative Visual Analytics

In Chapter 5 and Chapter 7, we proposed proactive approach to visual analytics that blends automatic computations with manual visual exploration, thus establishing a partnership between the analyst and the computer. This is the first step towards a fully proactive visualization system involving a human in the loop. Many improvements can be made towards a more efficient system. One potential future research topic is guiding recommendations based on the analyst's interest, past interactions, and even their personality. We presented pioneer work in Chapter 7 where navigational behavior has been taken into consideration. We intend to explore the Sherpa model further, including applying it to new datasets and computations, and investigating additional implicit computational steering mechanisms beyond navigation. We are also interested in studying how mining attention using Sherpa can be best realized for a team of analysts exploring a dataset synchronously. Other ideas may include figuring out the analyst's click stream, browsing and analysis history, and even social media profiles to determine how to best guide the proactive computation.

Appendix A: DataSite User Study Protocol

This Appendix contains the procedure and exit survey in the user study conducted in Chapter 5.

A.1 Procedure

A.1.1 Purpose of the Study

- **Primary purpose**: Evaluation of the impact of Feed (automatically generated insight) on users exploration of the dataset.
- **Secondary purpose**: Comparative evaluation of DataSite and PoleStar(a Tableau-like manual visual specification tool).

A.1.2 Introduction

- Experimenter welcomes subject.
- Experimenter tell the participants that we are using a data visualization tools to explore the datasets.
- Start the interface in INCOGNITON window.

- Complete pre-study questionnaire (attached later).

A.1.3 Study Session 1

- Practice/training for PoleStar.
- Start audio recording and screen recording.
- Refresh the window.
- Task explanation: You are given the tool and one dataset, your task is to explore and understand the dataset as much as possible. You are given 20 minutes to explore. Whenever you have some insights about the dataset, feel free to speak out aloud about what you have found. You are encouraged to find more complicated observations involving two or more attributes, such as "Cars from USA with 4 cylinders are the most frequent item".
- (After the study) Complete post-study questionnaire for PoleStar.

A.1.4 Study Session 2

- Practice/training for DataSite.
- Start audio recording and screen recording.
- Refresh the window.
- Task explanation: You are given the tool and one dataset, your task is to explore and understand the dataset as much as possible. You are given 20

minutes to explore. Whenever you have some insights about the dataset, feel free to speak out aloud about what you have found. You are encouraged to find more complicated observations involving two or more attributes, such as "Cars from USA with 4 cylinders are the most frequent item".

- Complete post-study questionnaire for DataSite.

A.1.5 Exit Survey and Comments

- Complete post-study questionnaire comparing the two tools.
- Ask the participant to give any feedback or comments to the tools/study.
- Thanks the participants and pay them.

A.2 Sample Questions

Some leading questions for user study. Take Movies Dataset (7 categorical attributes, 8 numerical attributes) as an example.

A.2.1 Simple questions

- Which distributor has the largest amount of films produced?
- Which director has the largest amount of films produced?
- Which distributor/creative type/director has the largest amount of films produced?

- Which range of IMDB rating/rotten tomato ratings/ Production budget has the largest number of counts/films?

A.2.2 Medium Questions

- Which genre of movies/director has the largest worldwide gross/production budget in average?
- Which genre of movies/distributors has the highest IMDB ratings/rotten tomato rating?
- Is there any relationship between rotten tomatoes and IMDB ratings? If so, what is the relationship and how?
- Does US gross have any effect on worldwide gross? What about DVD sales?
- Does Running time in minutes have any effects on the IMDB votes?
- Does Running time in minutes have any relationship with the IMDB rates?
- Does Running time in minutes have any relationship with the Production budget?
- Does Running time in minutes have any relationship with the USDVD sales?
- Which director has the highest votes in IMDB?
- Which film(title) has the highest US gross/IMDB rating/rotten tomato rating?
- How many films have more than 250M production budget?

A.2.3 Difficult Questions

- Among all action movies, which action (major genre) movies(title) have the highest production budget?
- Among all action movies, which action (major genre) movies have the highest IMDB rating/rotten tomatoes rating?
- Among all action movies, which action (major genre) movies have the largest US/worldwide gross?
- Among all creative types of movies, which one has the largest IMDB Votes/Ratings?

A.3 Pre-Study Questionaire

1. Gender
   - (A) Male
   - (B) Female
2. Age
   - (A) 18-25
   - (B) 26-35
   - (C) 36-45
   - (D) 46 and over

3. Experience using data analysis and visualization tools(check all that apply).

   - Excel
   - Tableau
   - Python/matplotlib/etc packages
   - R/ggplot
   - I have not done any data analysis before
   - Other (list your tool):

4. Years of experience in data analysis/visualizations

   (A) less than 1 year

   (B) 1 - 2 years

   (C) 2 - 3 years

   (D) more than 3 years

   (E) No experience

A.3.1 PoleStar Exit Survey

1. **Efficiency**: Does the PoleStar interface provide an efficient way to find insig-hts from the data? (from 1 (least efficient) to 5 (most efficient))

2. **Ease of use**: Does the interface make it easy to find insights from the data? (from 1 (most difficult) to 5 (easiest))

3. **Enjoyability**: Is Polestar interface enjoyable and fun to work with? (from 1 (not at all) to 5 (very enjoyable))

4. How do you feel the comprehensiveness of your analysis with PoleStar? (from 1 (totally incomprehensive) to 5 (totally comprehensive))

A.3.2 DataSite Exit Survey

1. **Efficiency**: Does the DataSite interface provide an efficient way to find insights from the data? (from 1 (least efficient) to 5 (most efficient))

2. **Ease of use**: Does the interface make it easy to find insights from the data? (from 1 (most difficult) to 5 (easiest))

3. **Enjoyability**: Is DataSite interface enjoyable and fun to work with? (from 1 (not at all) to 5 (very enjoyable))

4. How do you feel the comprehensiveness of your analysis with dataSite? (from 1 (totally incomprehensive) to 5 (totally comprehensive))

5. Does the summaries in the Feed give you a guidance of your data analysis? (from 1 (absolutely not, it bothers me) to 5 (absolutely))

A.3.3 Exit Survey for Comparing Two Systems

1. Which one is more valuable for data exploration (finding insight of the data)? From 1 (PoleStar is more valuable) to 5 (DataSite is more valuable)

2. Which one is more valuable for focused question answering (e.g., given a specific question of the dataset to answer)? From 1 (PoleStar is more valuable) to 5 (DataSite is more valuable)

3. Do you think Feed is a useful field in data analysis? From 1 (Not useful at all) to 5 (very useful)

4. Suppose you are given a question regarding the dataset, will you use Feed to scan through/search for the answers? From 1 (Impossible) to 5 (Most probably)

5. Do you think the dynamically update results in the Feed in DataSite(just like Facebook Feed, or Twitter) will assist you in exploring the dataset? From 1 (Not at all) to 5 (Pretty sure)

6. Have you used PoleStar or DataSite before?

   (A) Yes

   (B) No

7. How do you think of the PoleStar/DataSite systems? Could you please give any comments? You can just speak out!

Bibliography

[1] S Mills, S Lucas, L Irakliotis, M Rappa, T Carlson, and B Perlowitz. Demystifying big data: a practical guide to transforming the business of government. *TechAmerica Foundation, Washington*, 2012.

[2] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: from big data to big impact. *MIS quarterly*, pages 1165–1188, 2012.

[3] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data.* McGraw-Hill Osborne Media, 2011.

[4] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Gorg, Jorn Kohlhammer, and Guy Melançon. Visual analytics: Definition, process, and challenges. *Information Visualization*, 4950:154–176.

[5] Pak Chung Wong and Jim Thomas. Visual analytics. *IEEE Computer Graphics and Applications*, (5):20–21, 2004.

[6] John W. Tukey. *Exploratory Data Analysis.* Pearson, Reading, MA, 1977.

[7] Kristin A Cook and James J Thomas. Illuminating the path: The research and development agenda for visual analytics. *IEEE Computer Society*, 2005.

[8] James J. Thomas and Kristin A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics.* IEEE Computer Society Press, 2005.

[9] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[10] Max Grusky, Jeiran Jahani, Josh Schwartz, Dan Valente, Yoav Artzi, and Mor Naaman. Modeling sub-document attention using viewport time. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 6475–6480, New York, NY, USA, 2017. ACM.

[11] E Wes Bethel, Hank Childs, and Charles Hansen. *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, 2012.

[12] Sriram Karthik Badam, Jieqiong Zhao, Shivalik Sen, Niklas Elmqvist, and David Ebert. TimeFork: Interactive prediction of time series. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 5409–5420, 2016.

[13] Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, pages 15–22, 2004.

[14] Patrick Baudisch and Christian Holz. My new pc is a mobile phone. *XRDS: Crossroads, The ACM Magazine for Students*, 16(4):36–41, 2010.

[15] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of ACM Conference on Research and Development in Information Retrieval*, pages 253–260, 2002.

[16] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.

[17] Randall M Rohrer and Edward Swing. Web-based information visualization. *IEEE Computer Graphics & Applications*, 17(4):52–59, 1997.

[18] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, 2009.

[19] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[20] Gonzalo J. Martínez and Leonardo Val. Implementing crossplatform distributed algorithms using standard web technologies. In *Proceedings of the Latin American Computing Conference (CLEI)*, pages 1–8, 2014.

[21] Gonzalo J. Martínez and Leonardo Val. Capataz: a framework for distributing algorithms via the world wide web. *CLEI Electronic Journal*, 18(2):1, 2015.

[22] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. imMens: Real-time visual querying of big data. *Computer Graphics Forum*, 32(3pt4):421–430, 2013.

[23] Enrico Bertini and Giuseppe Santucci. Give chance a chance: modeling density to enhance scatter plot quality through random data sampling. *Information Visualization*, 5(2):95–110, 2006.

[24] Anish Das Sarma, Hongrae Lee, Hector Gonzalez, Jayant Madhavan, and Alon Halevy. Efficient spatial sampling of large geographical tables. In *Proceedings of the ACM Conference on Management of Data*, pages 193–204, 2012.

[25] James P Bagrow, Erik M Bollt, Joseph D Skufca, and Daniel Ben-Avraham. Portraits of complex networks. *Europhysics Letters*, 81(6):68004, 2008.

[26] Ming C Hao, Umeshwar Dayal, Ratnesh K Sharma, Daniel A Keim, and Halldór Janetzko. Visual analytics of large multidimensional data using variable binned scatter plots. In *Proceedings of IS&T/SPIE Electronic Imaging*, pages 753006–753006, 2010.

[27] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the ACM Conference on Advanced Visual Interfaces*, pages 547–554, 2012.

[28] Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 313–317, 1994.

[29] Adrian Mayorga and Michael Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1526–1538, 2013.

[30] Niklas Elmqvist and Jean-Daniel Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454, 2010.

[31] Anish Das Sarma, Hongrae Lee, Hector Gonzalez, Jayant Madhavan, and Alon Halevy. Efficient spatial sampling of large geographical tables. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 193–204, 2012.

[32] Daniel B Carr, Richard J Littlefield, WL Nicholson, and JS Littlefield. Scatterplot matrix techniques for large N. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[33] 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 *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 1673–1682, New York, NY, USA, 2012. ACM.

[34] Pak Chung Wong, Han-Wei Shen, Christopher R Johnson, Chaomei Chen, and Robert B Ross. The top 10 challenges in extreme-scale visual analytics. *IEEE Computer Graphics & Applications*, 32(4):63, 2012.

[35] Chad A Steed, Daniel M Ricciuto, Galen Shipman, Brian Smith, Peter E Thornton, Dali Wang, Xiaoying Shi, and Dean N Williams. Big data visual analytics for exploratory Earth system simulation analysis. *Computers & Geosciences*, 61:71–82, 2013.

[36] Lauro Didier Lins, James T. Klosowski, and Carlos Eduardo Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[37] Jaegul Choo and Haesun Park. Customizing computational methods for visual analytics with big data. *IEEE Computer Graphics & Applications*, 33(4):22–28, 2013.

[38] Charles D. Stolper, Adam Perer, and David Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014.

[39] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[40] Daniel A Keim and H-P Kriegel. Visdb: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49, 1994.

[41] Mark Derthick, John Kolojejchick, and Steven F Roth. An interactive visualization environment for data exploration. In *Proceedings of Knowledge Discovery in Databases*, pages 2–9, 1997.

[42] Leilani Battle, Remco Chang, and Michael Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1363–1375, 2016.

[43] Danyel Fisher. Incremental, approximate database queries and uncertainty for exploratory visualization. In *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization*, pages 73–80, 2011.

[44] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. Distributed and interactive cube exploration. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 472–483, 2014.

[45] Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

[46] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[47] Yong Wang and Min S Kim. Bandwidth-adaptive clustering for mobile ad hoc networks. In *Proceedings of the IEEE Conference on Computer Communications and Networks*, pages 103–108, 2007.

[48] Seungbae Lee, Kanika Grover, and Alvin Lim. Enabling actionable analytics for mobile devices: performance issues of distributed analytics on Hadoop mobile clusters. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):15, 2013.

[49] Sachin Goyal and John Carter. A lightweight secure cyber foraging infrastructure for resource-constrained device. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 186–195, 2004.

[50] Shih-Hao Hung, Chi-Sheng Shih, Jeng-Peng Shieh, Chen-Pang Lee, and Yi-Hsiang Huang. Executing mobile applications on the cloud: Framework and issues. *Computers & Mathematics with Applications*, 63(2):573–587, 2012.

[51] Muhammad Shiraz, Mehdi Sookhak, Abdullah Gani, and Syed Adeel Ali Shah. A study on the critical analysis of computational offloading frameworks for mobile cloud computing. *Journal of Network and Computer Applications*, 47:47–60, 2015.

[52] Mohammed Anowarul Hassan and Songqing Chen. Mobile MapReduce: Minimizing response time of computing intensive mobile applications. In *Mobile Computing, Applications, and Services*, pages 41–59. 2012.

[53] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79. 2012.

[54] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind, and Hartmut Ziegler. Challenges in visual data analysis. In *Proceedings of the International Conference on Information Visualization*, pages 9–16, 2006.

[55] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, 1994.

[56] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 401–408, 2003.

[57] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[58] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986.

[59] Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps.* University of Wisconsin Press, 1983.

[60] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[61] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, 2007.

[62] Steven F. Roth, John Kolojejchick, Joe Mattis, and Jade Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 112–117, 1994.

[63] Jinwook Seo and Ben Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization*, 4(2):96–113, 2005.

[64] Manasi Vartak, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. SeeDB: Automatically generating query visualizations. *Proceedings of the VLDB Endowment*, 7(13):1581–1584, 2014.

[65] Daniel B. Perry, Bill Howe, Alicia M. F. Key, and Cecilia Aragon. VizDeck: Streamlining exploratory visual analytics of scientific data. In *iConference Proceedings*, pages 338–350, 2013.

[66] Stef van den Elzen and Jarke J. van Wijk. Small multiples, large singles: A new approach for visual data exploration. In *Computer Graphics Forum*, volume 32, pages 191–200, 2013.

[67] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016.

[68] Bahador Saket, Hannah Kim, Eli T. Brown, and Alex Endert. Visualization by demonstration: An interaction paradigm for visual data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):331–340, 2017.

[69] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *Proceedings of the Very Large Database Endowment*, 10(4):457–468, 2016.

[70] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 2648–2659, 2017.

[71] Renato Novais, Camila Nunes, Caio Lima, Elder Cirilo, Francisco Dantas, Alessandro Garcia, and Manoel Mendonça. On the proactive and interactive visualization for feature evolution comprehension: An industrial investigation. In *Proceedings of the International Conference on Software Engineering*, pages 1044–1053, 2012.

[72] John Alexis Guerra Gómez, Audra Buck-Coleman, Catherine Plaisant, and Ben Shneiderman. Treeversity: Comparing tree structures by topology and node's attributes differences. In *Proceedings of the IEEE Conference on Visual Analytics Science & Technology*, pages 275–276, 2011.

[73] Kedar Dhamdhere, Kevin S. McCurley, Ralfi Nahmias, Mukund Sundararajan, and Qiqi Yan. Analyza: Exploring data with conversation. In *Proceedings of ACM Conference on Intelligent User Interfaces*, pages 493–504, 2017.

[74] Robert van Liere, Jurriaan D. Mulder, and Jarke J. van Wijk. Computational steering. *Future Generation Computer Systems*, 12(5):441–450, April 1997.

[75] Jurriaan D. Mulder, Jarke J. van Wijk, and Robert van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15(1):119–129, February 1999.

[76] Steven G. Parker and Christopher R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, Piscataway, NJ, USA, 1995. IEEE.

[77] Jeffrey Vetter and Karsten Schwan. Progress: A toolkit for interactive program steering. In *Proceedings of the International Conference on Parallel Processing*, pages 139–142, Boca Raton, USA, 1995. CRC Press.

[78] Jeffrey Vetter and Karsten Schwan. High performance computational steering of physical simulations. In *Proceedings of the International Parallel Processing Symposium*, pages 128–132, Piscataway, NJ, USA, 1997. IEEE.

[79] David J. Jablonowski, John D. Bruner, Brian Bliss, and Robert B. Haber. VASE: The visualization and application steering environment. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 560–569, Piscataway, NJ, USA, 1993. IEEE.

[80] Helmut Doleisch, Helwig Hauser, Martin Gasser, and Robert Kosara. Interactive focus+context analysis of large, time-dependent flow simulation data. *Simulation*, 82(12):851–865, 2006.

[81] John Biddiscombe, Jerome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. Parallel computational steering and analysis for HPC applications using a ParaView interface and the HDF5 DSM virtual file driver. In *Proceedings of the Eurographics Conference on Parallel Graphics*

and Visualization, pages 91–100, Geneva, Switzerland, 2011. Eurographics Association.

[82] Jürgen Waser, Raphael Fuchs, Hrvoje Ribicic, Benjamin Schindler, Gunther Bloschl, and Eduard Gröller. World lines. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1458–1467, 2010.

[83] Jürgen Waser, Hrvoje Ribicic, Raphael Fuchs, Christian Hirsch, Benjamin Schindler, Gunther Bloschl, and Eduard Gröller. Nodes on ropes: A comprehensive data and control flow for steering ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1872–1881, 2011.

[84] Hrvoje Ribicic, Jürgen Waser, Raphael Fuchs, Guenter Bloschl, and Eduard Gröller. Visual analysis and steering of flooding simulations. *IEEE Transactions on Visualization and Computer Graphics*, 19(6):1062–1075, 2013.

[85] Sungahn Ko, Jieqiong Zhao, Jing Xia, Shehzad Afzal, Xiaoyu Wang, Greg Abram, Niklas Elmqvist, Len Kne, David Van Riper, Kelly P. Gaither, Shaun Kennedy, William J. Tolone, William Ribarsky, and David S. Ebert. VASA: Interactive computational steering of large asynchronous simulation pipelines for societal infrastructure. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1853–1862, 2014.

[86] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2122–2131, 2014.

[87] Danyel Fisher, Robert DeLine, Mary Czerwinski, and Steven M. Drucker. Interactions with big data analytics. *Interactions*, 19(3):50–59, 2012.

[88] Sriram Karthik Badam, Niklas Elmqvist, and Jean-Daniel Fekete. Steering the craft: Ui elements and visualizations for supporting progressive visual analytics. *Computer Graphics Forum*, 36(3):491–502, 2017.

[89] Emanuel Zgraggen, Alex Galakatos, Andrew Crotty, Jean-Daniel Fekete, and Tim Kraska. How progressive visualizations affect exploratory analysis. *IEEE Transactions on Visualization and Computer Graphics*, 23(8):1977–1987, 2017.

[90] Jean-Daniel Fekete. ProgressiVis: a toolkit for steerable progressive analytics and visualization. In *Proceedings of the IEEE VIS Workshop on Data Systems for Interactive Analysis*, page 5, 2015.

[91] Jaemin Jo, Jinwook Seo, and Jean-Daniel Fekete. PANENE: A progressive algorithm for indexing and querying approximate k-nearest neighbor. *IEEE Transactions on Visualization and Computer Graphics*, PP(1):1–14, 2018. To appear.

[92] Nicola Pezzotti, Boudewijn P. F. Lelieveldt, Laurens van der Maaten, Thomas Hollt, Elmar Eisemann, and Anna Vilanova. Approximated and user steerable tSNE for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1739–1752, July 2017.

[93] Eric Bier, Maureen Stone, and Ken Pier. Enhanced illustration using magic lens filters. *IEEE Computer Graphics and Applications*, 17(6):62–70, November/December 1997.

[94] W3C. Scalable Vector Graphics (SVG) 1.0 specification, Apr 2001.

[95] Erik Dahlstrøm, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathan Watt, Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. Scalable Vector Graphics (SVG) 1.1 (second edition), 2011.

[96] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive Vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, 2016.

[97] Geoffrey Ellis and Alan Dix. Enabling automatic clutter reduction in parallel coordinate plots. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):717–724, Sept 2006.

[98] Ed H Chi. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 69–75, 2000.

[99] Deokgun Park, Steven M. Drucker, Roland Fernandez, and Niklas Elmqvist. ATOM: A grammar for unit visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 2018. To appear.

[100] Ben Shneiderman. Extreme visualization: squeezing a billion records into a million pixels. In *Proceedings of the ACM Conference on Management of Data*, pages 3–12, 2008.

[101] Matthias Nielsen, Niklas Elmqvist, and Kaj Grønbæk. Scribble query: Fluid touch brushing for multivariate data visualization. In *Proceedings of the 28th Australian Conference on Computer-Human Interaction*, OzCHI '16, pages 381–390, 2016.

[102] Aritra Dasgupta and Robert Kosara. Pargnostics: Screen-space metrics for parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1017–1026, Nov 2010.

[103] Julian Heinrich and Daniel Weiskopf. State of the art of parallel coordinates. In *Eurographics – State of the Art Reports*. The Eurographics Association, 2013.

[104] Enrico Bertini and Giuseppe Santucci. Quality metrics for 2D scatterplot graphics: Automatically reducing visual clutter. In *Smart Graphics*, volume 3031 of *Lecture Notes in Computer Science*, pages 77–89. Springer, 2004.

[105] Matej Novotný and Helwig Hauser. Outlier-preserving focus+context visualization in parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):893–900, Sept 2006.

[106] Zhe Cui, Shivalik Sen, Sriram Karthik Badam, and Niklas Elmqvist. Vishive: Supporting web-based visualization through ad hoc computational clusters of mobile devices. *Information Visualization*.

[107] Selan Dos Santos and Ken Brodlie. Gaining understanding of multivariate and multidimensional data through visualization. *Computers & Graphics*, 28(3):311–325, 2004.

[108] Fernanda B Viégas, Martin Wattenberg, Frank Van Ham, Jesse Kriss, and Matt McKeon. ManyEyes: A site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.

[109] Sriram Karthik Badam, Eli Fisher, and Niklas Elmqvist. Munin: A peer-to-peer middleware for ubiquitous analytics and visualization spaces. *IEEE Transactions on Visualization and Computer Graphics*, 21(2):215–228, Feb 2015.

[110] Eli Raymond Fisher, Sriram-Karthik Badam, and Niklas Elmqvist. Designing peer-to-peer distributed user interfaces: Case studies on building distributed applications. *International Journal of Human-Computer Studies*, 72(1):100–110, 2014.

[111] Peter Doemel. WebMap: a graphical hypertext navigation tool. *Computer Networks and ISDN Systems*, 28(1):85–97, 1995.

[112] Raphael Fuchs, Jürgen Waser, and Meister Eduard Gröller. Visual human+ machine learning. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1327–1334, 2009.

[113] Abish Malik, Ross Maciejewski, Yun Jang, Whitney Huang, Niklas Elmqvist, and David Ebert. A correlative analysis process in a visual analytics environment. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*, pages 33–42, 2012.

[114] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[115] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *IEEE symposium on Mass storage systems and technologies (MSST)*, pages 1–10, 2010.

[116] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2), 2008.

[117] Patrick Baudisch and Christian Holz. My new PC is a mobile phone. *ACM Crossroads*, 16(4):36–41, 2010.

[118] Mike Bostock. Thinking with Joins. `https://bost.ocks.org/mike/join/`, accessed Oct 2017.

[119] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Journal of Information Processing and Management*, 24(5):513–523, 1988.

[120] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[121] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[122] Fernando Pérez and Brian E Granger. IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, 2007.

[123] Zhe Cui, Sriram Karthik Badam, Adil Yalçin, and Niklas Elmqvist. DataSite: Proactive visual data exploration with computation of insight-based recommendations. *Information Visualization*, 2018.

[124] PoleStar, 2017. `https://vega.github.io/polestar/`.

[125] Andrea Batch and Niklas Elmqvist. The interactive visualization gap in initial exploratory data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[126] Karl Pearson. Notes on regression and inheritance in the case of two parents. In *Proceedings of the Royal Society of London*, volume 58, pages 240–242, 1895.

[127] Harold V. Henderson and Paul F. Velleman. Building multiple regression models interactively. *Biometrics*, pages 391–411, 1981.

[128] Ernesto Ramos and David Donoho. Asa data exposition dataset. *CMU Dataset Archive*, 1983.

[129] Dale J. Barr, Roger Levy, Christoph Scheepers, and Harry J. Tily. Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, 68(3):255–278, 2013.

[130] Spence Green, Jeffrey Heer, and Christopher D. Manning. The efficacy of human post-editing for language translation. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 439–448, 2013.

[131] Justin Wagner, Florin Chelaru, Jayaram Kancherla, Joseph N Paulson, Alexander Zhang, Victor Felix, Anup Mahurkar, Niklas Elmqvist, and Hctor CorradaBravo. Metaviz: interactive statistical and visual analysis of metagenomic data. *Nucleic Acids Research*, 2018.

[132] Florin Chelaru, Llewellyn Smith, Naomi Goldstein, and Héctor Corrada Bravo. Epiviz: interactive visual analytics for functional genomics data. *Nature Methods*, 11(9):938–940, 2014.

[133] Winston Timp, Hector C. Bravo, Oliver G. McDonald, Michael Goggins, Chris Umbricht, Martha Zeiger, Andrew P. Feinberg, and Rafael A. Irizarry. Large hypomethylated blocks as a universal defining epigenetic alteration in human solid tumors. *Genome Medicine*, 6(8), 2014.

[134] Martin J. Aryee, Andrew E. Jaffe, Hector Corrada-Bravo, Christine Ladd-Acosta, Andrew P. Feinberg, Kasper D. Hansen, and Rafael A. Irizarry. Minfi: A flexible and comprehensive bioconductor package for the analysis of infinium dna methylation microarrays. *Bioinformatics*, 30(10):1363–1369, 2014.

[135] Jayaram Kancherla, Alexander Zhang, Brian Gottfried, and Hector Corrada Bravo. Epiviz web components: reusable and extensible component library to visualize functional genomic datasets. *F1000Research*, 7, 2018.

[136] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[137] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. [Online; accessed ¡today¿].

[138] Ian Fette. The websocket protocol. 2011.

[139] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[140] Matthew N McCall, Harris A Jaffee, Susan J Zelisko, Neeraj Sinha, Guido Hooiveld, Rafael A Irizarry, and Michael J Zilliox. The gene expression barcode 3.0: Improved data processing and mining tools. *Nucleic Acids Research*, 42(D1):D938–D943, 2013.

[141] Winston Timp, Hector Corrada Bravo, Oliver G McDonald, Michael Goggins, Chris Umbricht, Martha Zeiger, Andrew P Feinberg, and Rafael A Irizarry. Large hypomethylated blocks as a universal defining epigenetic alteration in human solid tumors. *Genome medicine*, 6(8):61, 2014.

[142] Graham JG Upton. Fisher's exact test. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, pages 395–402, 1992.

[143] D Craig Allred, Robert W Carlson, Donald A Berry, Harold J Burstein, Stephen B Edge, Lori J Goldstein, Allen Gown, M Elizabeth Hammond, James Dirk Iglehart, Susan Moench, et al. Nccn task force report: estrogen receptor and progesterone receptor testing in breast cancer by immunohistochemistry. *Journal of the National Comprehensive Cancer Network*, 7(Suppl 6):S–1, 2009.

[144] Christina Davies, Hongchao Pan, Jon Godwin, Richard Gray, Rodrigo Arriagada, Vinod Raina, Mirta Abraham, Victor Hugo Medeiros Alencar, Atef Badran, Xavier Bonfill, et al. Long-term effects of continuing adjuvant tamoxifen to 10 years versus stopping at 5 years after diagnosis of oestrogen receptor-positive breast cancer: Atlas, a randomised trial. *The Lancet*, 381(9869):805–816, 2013.

[145] Early Breast Cancer Trialists' Collaborative Group et al. Relevance of breast cancer hormone receptors and other factors to the efficacy of adjuvant tamoxifen: patient-level meta-analysis of randomised trials. *The lancet*, 378(9793):771–784, 2011.

[146] Irwin H Gelman. Emerging roles for ssecks/gravin/akap12 in the control of cell proliferation, cancer malignancy, and barriergenesis. *Genes & cancer*, 1(11):1147–1156, 2010.

[147] Bartosz Kazimierz Słowikowski, Margarita Lianeri, and Paweł Piotr Jagodziński. Exploring estrogenic activity in lung cancer. *Molecular biology reports*, 44(1):35–50, 2017.

[148] Tiago C Silva, Simon G Coetzee, Nicole Gull, Lijing Yao, Dennis J Hazelett, Houtan Noushmehr, De-Chen Lin, and Benjamin P Berman. Elmer v. 2: an r/bioconductor package to reconstruct gene regulatory networks from dna methylation and transcriptome profiles. *Bioinformatics*.

[149] Jeffrey S. Vetter. Computational steering annotated bibliography. *SIGPLAN Notices*, 32(6):40–44, 1997.

[150] Polymer. Polymer, 2018. `https://www.polymer-project.org/`.

[151] Epiviz. Epiviz data provider, 2018.

[152] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of the International Conference on Intelligence Analysis*, volume 5, pages 2–4, McLean, VA, USA, 2005. The MITRE Corporation.

[153] Boris M. Velichkovsky, Andreas Sprenger, and Pieter Unema. Towards gaze-mediated interaction: Collecting solutions of the ”midas touch problem”. In *Proceedings of the INTERACT Conference*, pages 509–516, Boston, MA, USA, 1997. Springer.

[154] David Kirsh and Paul P. Maglio. On distinguishing epistemic from pragmatic action. *Cognitive Science*, 18(4):513–549, 1994.