VisR: An interactive visualization framework for analysis of sequencing data

by

Hamid Younesy

M.Sc., Simon Fraser University, 2005
B.Sc., Sharif University of Technology, 2002

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

in the

School of Computing Science
Faculty of Applied Sciences

© Hamid Younesy 2019
SIMON FRASER UNIVERSITY
Fall 2019

Copyright in this work rests with the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.
# Approval

**Name:** Hamid Younesy  
**Degree:** Doctor of Philosophy  
**Title:** VisR: An interactive visualization framework for analysis of sequencing data  
**Examining Committee:**  
- **Chair:** Steven Bergner  
  University Research Associate  
- **Richard (Hao) Zhang**  
  Senior Supervisor  
  Professor  
- **Torsten Möller**  
  Co-supervisor  
  Adjunct Professor  
- **Martin Ester**  
  Supervisor  
  Professor  
- **Maxwell Libbrecht**  
  Internal Examiner  
  Assistant Professor  
- **Michael Gleicher**  
  External Examiner  
  Professor  
  Department of Computer Sciences  
  University of Wisconsin, Madison  

**Date Defended:** October 23, 2019
Ethics Statement

The author, whose name appears on the title page of this work, has obtained, for the research described in this work, either:

a. human research ethics approval from the Simon Fraser University Office of Research Ethics

or

b. advance approval of the animal care protocol from the University Animal Care Committee of Simon Fraser University

or has conducted the research

c. as a co-investigator, collaborator, or research assistant in a research project approved in advance.

A copy of the approval letter has been filed with the Theses Office of the University Library at the time of submission of this thesis or project.

The original application for approval and letter of approval are filed with the relevant offices. Inquiries may be directed to those authorities.

Simon Fraser University Library
Burnaby, British Columbia, Canada

Update Spring 2016
Abstract

Several tools have been developed to enable biologists to perform initial browsing and exploration of sequencing data. However, the computational tool set for further analyses often requires significant computational expertise to use and many of the biologists with the knowledge needed to interpret these data must rely on programming experts. In this thesis, we focus on addressing this limitation through visualization tools for exploratory analysis of sequencing data and contribute the design and development of two novel systems that are flexible enough to allow a high degree of analysis power, while at the same time are easy to use for non-programmers: (1) a general purpose framework that bridges the gap between the biologists and the bioinformaticians through a system of visual analysis modules that can be rapidly developed and connected together, and (2) a first-of-its-kind system that facilitates visual parameter space analysis for a wide variety of computer models.

We start by providing a characterization of the data and an abstraction of the domain tasks in the field of epigenetics and present a design study on development and evaluation of ChAsE, an interactive tool to facilitate analysis and visualization of epigenetic datasets. We will then discuss VisR, a general framework for analysis of sequencing datasets that provides both a computationally rich as well as accessible framework for integrative and interactive analyses through modules called R-apps that utilize packages in R and repositories such as Bioconductor. Our framework provides means for interactive exploration of the results or the R-apps, and supports linking apps to create more complex workflows. It also provides an ecosystem to allow extension and sharing of the apps. We finally present ModEx, a general purpose system for exploring parameters of a variety of computer models. We discuss how the system offers key components of visual parameter space analysis frameworks including parameter sampling, deriving output summaries, and an interactive and customizable exploration interface and explain how it can be used for rapid development of custom solutions for different application domains.

**Keywords:** interactive visualization; sequencing data; epigenetics; R-project; parameter exploration; bioinformatics
Acknowledgements

This dissertation is the result of several years of collaborative effort and would not have been possible without the help, support and guidance of many people.

First and foremost, I would like to thank my supervisors. In particular, as it relates to this work, I am specially thankful to my supervisor Dr Torsten Möller who has been not only a mentor, but also a wonderful colleague and friend. Torsten’s knowledge of visualization and his inspiring and fresh ideas in presenting complicated topics helped make my work much more accessible and understandable to both the visualization audience and the genome sciences audience. I also want to acknowledge the excellent support I received from my supervisors Dr Richard (Hao) Zhang and Dr Martin Ester during this process.

I am grateful to our many collaborators for their patience, time, and interest in our projects. Without their support and insightful and valuable feedback this work would not have been possible. In particular I wanted to acknowledge Dr Mohammad Mahdi Karimi from MRC London Institute of Medical Sciences for providing tremendous guidance in understanding the biology and bioinformatics background as well as helping to connect me with several collaborators at the Genome Sciences Centre, Life Sciences Centre of UBC, and UBC Biomedical Research Centre, where I performed a great deal of my PhD research. I would also like to thank Dr Steven Jones from the Genome Sciences Centre for his valuable feedback during the development of VisR as well as providing research funding for my work. In addition, I would like to thank Dr Cydney Nielsen, Dr Olivia Alder, and Rebecca Cullum from BC Cancer Research Centre and Dr Brad Hoffman from BC Children’s Hospital Research Institute for the collaboration on the ChAsE project; Dr Matthew C. Lorincz, Julie Brind’Amour, Carol Chen, and Julien Richard Albert from Life Sciences Centre of UBC, and Dr Michael Underhill, Dr Fabio Rossi, Ryan Vander Werff, Dr Wilder Scott, and Elena Groppa from UBC Biomedical Research Centre for their feedback and help on development of the VisR and ModEx projects.

I would like to acknowledge Fan (Helena) Xu, Kathy Cheng, and Bernie (Yiwei) Zhao for contributing several new and improved R-apps to the VisR framework and Joseph Pober for his contributions to the development of ModEx.

I am extremely grateful to my colleagues in Koolhaus Games Inc., specifically Wolfgang Hamann, the Pres/CEO, and Haris Widjaya, technical project manager, for their support,
flexibility and understanding in providing the opportunity of working while completing my studies.

Finally, my deepest gratitude go to my family and friends in particular my parents and spiritual mentors whom I have always been blessed with their unconditional love and encouragement, and my wife Marzieh for all her support and the sacrifices she has made. We are grateful to God for blessing us with our son, Mohammad Hossein, and daughters, Fatemeh and Zahra, who have been an enormous source of inspiration and hope and have shown a great deal of patience and understanding for a busy dad.
Table of Contents

Approval ii
Ethics Statement iii
Abstract iv
Acknowledgements v
Table of Contents vi
List of Tables xi
List of Figures xii

1 Motivation 1
    1.1 Contribution .................................................. 2
    1.2 Organization of the thesis .................................... 3

2 Background and Related Work 4
    2.1 Biological background ......................................... 4
        2.1.1 Genome ..................................................... 4
        2.1.2 DNA sequencing technology ............................... 5
        2.1.3 Reference genome ......................................... 5
        2.1.4 Epigenetics ............................................... 5
    2.2 Sequencing data analysis ...................................... 6
        2.2.1 Data types and tasks ..................................... 6
        2.2.2 EDAM ontology ............................................ 8
        2.2.3 File formats ............................................. 9
    2.3 A primer on sequencing data visualization .................. 11
        2.3.1 Genomic coordinates ..................................... 11
        2.3.2 Genomic layouts .......................................... 12
        2.3.3 Genome browsers ......................................... 13
3 An interactive analysis and exploration tool for epigenomic data

3.1 Introduction

3.2 Biological background and data

3.2.1 Epigenetic marks

3.2.2 Region sets

3.2.3 Data abstraction

3.3 Task analysis

3.3.1 Task 1 - signal query

3.3.2 Task 2 - cluster

3.3.3 Task 3 - quality control

3.3.4 Task 4 - comparison

3.3.5 Task 5 - downstream analysis

3.4 Related work

3.5 ChAsE

3.5.1 Data input

3.5.2 Interface

3.5.3 Workspace Pane

3.5.4 Method panes

3.5.5 Heat Map Pane

3.5.6 Plot Pane

3.5.7 Favorites Pane

3.5.8 Common functionality

3.6 Evaluation

3.6.1 Case study 1: signal querying and clustering

3.6.2 Case study 2: exploration with the interactive heatmap

3.7 Lessons learned

3.8 Future work

4 A visual framework for analysis of sequencing data

4.1 Related work
4.1.1 R-based visualization systems ...................................... 60
4.1.2 Visualization systems for biological data analysis .......... 60
4.2 Requirement analysis .................................................... 61
  4.2.1 A typical analysis workflow ....................................... 61
  4.2.2 Design ............................................................... 62
4.3 The VisR framework ...................................................... 63
  4.3.1 VisR apps .......................................................... 63
  4.3.2 R-apps ............................................................... 64
  4.3.3 A simple R-app .................................................... 66
  4.3.4 Native apps ........................................................ 68
  4.3.5 The VisR graphical user interface ............................... 70
  4.3.6 Limitations .......................................................... 72
4.4 Evaluation ................................................................. 73
  4.4.1 Case study 1: gene expression in stages of mouse development ..... 73
  4.4.2 Case study 2: allele-specific gene expression .................. 76
4.5 Conclusions and future work ........................................... 77

5 A general purpose computer model exploration system 79
  5.1 Introduction ........................................................... 79
  5.2 Related work .......................................................... 81
    5.2.1 Applications in machine learning .............................. 81
    5.2.2 Applications in data mining .................................... 82
    5.2.3 Applications in bioinformatics ................................. 82
    5.2.4 Definitions ....................................................... 82
  5.3 A general purpose system for visual parameter space analysis .. 83
    5.3.1 Design goals ..................................................... 83
    5.3.2 Parameter sampling ............................................. 83
    5.3.3 ModEx app ....................................................... 86
  5.4 Evaluation .............................................................. 89
    5.4.1 Case study: clustering .......................................... 89
    5.4.2 Case study: neural network playground ....................... 95
    5.4.3 Case Study: differential gene expression ................... 101
  5.5 Conclusion and future work ......................................... 106
    5.5.1 Parameter sampling ............................................. 106
    5.5.2 Analyzing output ............................................... 107
    5.5.3 Browsing parameters ............................................ 107
    5.5.4 Guiding the exploration ....................................... 107

6 Conclusions ............................................................. 108
  6.1 Pitfalls of visual analysis systems ................................ 109
6.2 Future directions .................................................. 110
6.2.1 Provenance and reproducibility ............................. 110
6.2.2 Extending to other application domains .................... 110

Bibliography ......................................................... 112
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Sequencing data types and file formats.</td>
<td>10</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Visualization tools for quality control of sequencing data</td>
<td>24</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Tools for assembly visualization.</td>
<td>25</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>Motif visualization tools.</td>
<td>26</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>List of genome browsers.</td>
<td>28</td>
</tr>
<tr>
<td>Table 2.6</td>
<td>Visualization tools for variant analysis.</td>
<td>29</td>
</tr>
<tr>
<td>Table 2.7</td>
<td>Visualization tools for gene expression analysis.</td>
<td>31</td>
</tr>
<tr>
<td>Table 2.8</td>
<td>Visualization tools for epigenetics analysis.</td>
<td>32</td>
</tr>
<tr>
<td>Table 2.9</td>
<td>Visualization tools for single cell analysis.</td>
<td>34</td>
</tr>
<tr>
<td>Table 2.10</td>
<td>Tools for visualization of 3D genome structure</td>
<td>36</td>
</tr>
<tr>
<td>Table 2.11</td>
<td>Visual analysis frameworks for sequencing data.</td>
<td>37</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Supported types for input parameters</td>
<td>64</td>
</tr>
</tbody>
</table>
# List of Figures

<p>| Figure 2.1 | The sequencing data flow | 6 |
| Figure 2.2 | Visualization of some EDAM ontology concepts | 9 |
| Figure 2.3 | Examples of four common genomic layouts | 12 |
| Figure 2.4 | Genome browser track visualizations | 13 |
| Figure 2.5 | Example workflows created using visual analysis frameworks | 16 |
| Figure 2.6 | How surveys proliferate | 23 |
| Figure 2.7 | Quality control of raw sequencing reads | 24 |
| Figure 2.8 | Examples of sequence assembly visualization | 26 |
| Figure 2.9 | Examples of motif visualization | 27 |
| Figure 2.10 | Examples of variant visualization | 30 |
| Figure 2.11 | Example visualizations for gene expression analysis | 30 |
| Figure 2.12 | Example visualization for epigenetics | 33 |
| Figure 2.13 | Example visualization of single cell RNA-seq data | 34 |
| Figure 2.14 | Example visualizations of 3D genomic data | 35 |
| Figure 3.1 | Data input dialog | 44 |
| Figure 3.2 | ChAsE interface | 45 |
| Figure 3.3 | Profile plot views | 46 |
| Figure 3.4 | Method panes | 47 |
| Figure 3.5 | Sorting heatmaps by different metrics | 49 |
| Figure 3.6 | Using Comparison Pane | 52 |
| Figure 3.7 | Using Signal Query Pane | 53 |
| Figure 3.8 | $k$-means clustering following the signal query | 53 |
| Figure 3.9 | Heatmap sorted by average signal | 54 |
| Figure 3.10 | Clustering by H3K4me3 and 5-hmC into four clusters | 55 |
| Figure 3.11 | $k$-means clustering by H3K4me3 and H3K27me3 into three clusters | 56 |
| Figure 4.1 | Wire-frame prototype | 62 |
| Figure 4.2 | VisR evolution | 63 |
| Figure 4.3 | Components of SimplePlot R-app | 67 |
| Figure 4.4 | Using the R API to specify parameters for an R-app | 68 |
| Figure 4.5 | Interactive Apps | 69 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
<td>VisR interface</td>
<td>70</td>
</tr>
<tr>
<td>4.7</td>
<td>Workspace for gene expression analysis</td>
<td>75</td>
</tr>
<tr>
<td>4.8</td>
<td>Workspace for allele-specific gene expression analysis</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>The GUI generated from the parameter specification</td>
<td>84</td>
</tr>
<tr>
<td>5.2</td>
<td>The ModEx data flow</td>
<td>85</td>
</tr>
<tr>
<td>5.3</td>
<td>An overview of ModEx used in a clustering application domain</td>
<td>90</td>
</tr>
<tr>
<td>5.4</td>
<td>An overview of the Clustervision</td>
<td>92</td>
</tr>
<tr>
<td>5.5</td>
<td>Exploring the quality of the clusterings using different clustering quality metrics</td>
<td>93</td>
</tr>
<tr>
<td>5.6</td>
<td>Exploring top features of a selected clustering in Clustervision and ModEx</td>
<td>95</td>
</tr>
<tr>
<td>5.7</td>
<td>Visual analysis of the clusterings</td>
<td>96</td>
</tr>
<tr>
<td>5.8</td>
<td>Linked views in ModEx parameter space exploration</td>
<td>97</td>
</tr>
<tr>
<td>5.9</td>
<td>Selected plots from exploration of runs for the Playground case study</td>
<td>100</td>
</tr>
<tr>
<td>5.10</td>
<td>Patterns common in the runs for all datasets</td>
<td>101</td>
</tr>
<tr>
<td>5.11</td>
<td>R-apps for differential expression analysis</td>
<td>102</td>
</tr>
<tr>
<td>5.12</td>
<td>ModEx exploration of the DESeq app</td>
<td>104</td>
</tr>
<tr>
<td>5.13</td>
<td>ModEx exploration of the edgeR app</td>
<td>105</td>
</tr>
<tr>
<td>6.1</td>
<td>VisR at the front page of the CEEHRC website</td>
<td>109</td>
</tr>
</tbody>
</table>
Chapter 1

Motivation

The advancements in genome sequencing technology in the past two decades has revolutionized and transformed the approaches to many biological studies. Sequencing data is the generic name for the datasets acquired using high-throughput nucleic acid sequencing techniques. This technology can be used to measure the biochemical states of cells such as the expression levels of genes or binding sites of proteins in DNA. While computational methods to interpret these data continue to evolve and improve, there is great value in data exploration and many questions remain too ill-defined to be addressed in an automated fashion. Visualization is thus a valuable tool in this domain. In addition, the rapidly changing computational tool set for data analysis often requires significant computational expertise to use. Many of the biologists who possess the detailed knowledge needed to interpret these data must rely on programming experts. As a result, interactive visualization holds great promise in being able to lower the computational barrier to analysis and engage biology experts more directly in data processing and interpretation.

Sequencing data associate data with locations on the genome. Data can be categorical, like genome annotations, or numerical like gene activities. When looking at the software that researchers use for exploratory analysis of sequencing data, we can consider two aspects. First aspect is the usability, which is how easy can they be used specially by users without programming expertise. Second is how flexible are they to cover a broad range of basic to advanced functionality requirements.

General analysis tools like Microsoft Excel and Tableau are fairly easy to use and provide some statistical and analytical tools, but they provide minimum analysis power when it comes to sequencing data. There are also popular visualization tools called genome browsers which provide support for sequencing data but limited statistical and analytical capabilities. On the complementary side, there packages that are available through programming languages like R [1] and python and are hosted by repositories like Bioconductor [2]. These packages provide advanced methods for specific types of analysis, but they require programming as well as often significant time to get familiar with the usage documentation.
A common approach to address this gap is to provide a coding API. For example some general visual analysis tools and some genome browsers provide a programming API using R or python. There have also been efforts to take individual packages and create a graphical user interface for them using frameworks such as Shiny [3]. The graphical user interface helps making these packages easier to use for specific analysis tasks, but the development cost per tool is high and it is difficult to link them together. That is mostly because they have different user interfaces, and different requirements for platform and data formats.

So, there is this trade off between flexibility and usability in which “as the flexibility of a system increases, the usability of the system decreases”. Our goal was to reduce this trade-off and provide a solution that is flexible enough to allow a high degree of analysis power, while at the same time is easy to use.

1.1 Contribution

In this thesis, we focus on visualization tools for exploratory analysis of sequencing data. Our main two scientific contributions are the design and development of:

1. a general purpose framework that bridges the gap between the biologists and the bioinformaticians through a system of visual analysis modules that can be rapidly developed and connected together.

2. a first-of-its-kind system that provides visual parameter analysis for a wide variety of computer models.

However, the chronological order of our contributions is as follows. Our first project in sequencing data visualization was a collaboration with biologists on analysis of epigenomics datasets. We contribute the data and task abstractions for the analysis of epigenetic data, and the justified design and implementation of ChAsE, an interactive tool to facilitate data analysis and visualization in this domain.

During the development of ChAsE, we encountered several instances where we had to spend a great deal of time to implement features which already existed as command-line tools or libraries, but could not be easily integrated into our tool due to platform and architecture limitations. To address that, we developed VisR, a framework that provides a computationally rich and accessible platform for integrative and interactive analyses of sequencing data without requiring programming expertise. We achieved this aim by providing R-apps, which offer a semi auto-generated and unified graphical user interface for computational packages in R and a platform to connect the apps to create more complex workflows.

The rapid development API of VisR allowed us to create many R-apps and attract a growing number of users since the release. However, even though the graphical user interface
of the R-apps allowed non-programmers to utilize the computational power of the R environment, a new challenge surfaced. Most apps had several control parameters and finding the “right” parameters for each use case involved a tedious manual trial-and-error approach. Encouraged by the previous successful attempts with visual parameter exploration tools we implemented ModEx an extension of VisR to support general visual parameter space analysis for R-apps. To summarize, the main contributions of this thesis are:

- VisR, an integrated framework that provides: (a) tools for analysis of sequencing data, (b) a framework for rapid development of R modules called R-apps, (c) support for interactive exploration of the results, (d) support for linking apps to create more complex workflows, and (e) an ecosystem to allow extension and sharing of the apps.

- ModEx, which extends VisR to offer a general visual parameter space analysis framework that provides: (a) parameter sampling, (b) deriving output summaries, and (c) interactive and customizable exploration interface.

### 1.2 Organization of the thesis

The rest of the thesis is organized as follows:

In Chapter 2 we review the data, tasks, and visualization techniques for the analysis of sequencing data. We will provide a brief overview of the biological background and discuss the data types and tasks from raw sequencing reads to single-cell gene expression matrices. We will then review the existing surveys of the past years and finally present our survey of the state of the art methods for visualizing sequencing data.

In Chapter 3 we present a design study and discuss our analysis and abstraction of the data and tasks related to the domain of epigenomics and the design and implementation of ChAsE, an interactive tool to facilitate data analysis and visualization in this domain.

Chapter 4 describes a framework for analysis of sequencing datasets that provides both a computationally rich as well as accessible framework for integrative and interactive analyses. In this framework called VisR, modules called apps, offer a semi-auto generated and unified graphical user interface for computational packages in R and repositories such as Bioconductor.

In Chapter 5, we present a general purpose system that can be used for exploring parameters of a variety of computer models. Our proposed system called ModEx offers key components of parameter analysis frameworks including parameter sampling, deriving output summaries, and an exploration interface. It also provides an API for rapid development of parameter space exploration solutions as well as the flexibility to support custom workflows for different application domains.

Finally, in Chapter 6, we offer a summary and conclusion of our contributions, as well as a discussion of possible directions for future work.
Chapter 2

Background and Related Work

Sequencing data is the generic name for the datasets acquired using high-throughput nucleic acid sequencing techniques. This technology can be used to measure the biochemical states of cells such as the expression levels of genes or binding sites of proteins in DNA. In this chapter will review the data, tasks, and visualization methods for the analysis of sequencing data. We will start with a brief overview of the biological background in Section 2.1 and then will discuss the sequencing data analysis workflow including data types and tasks in Section 2.2 followed by a background on sequencing data visualization in Section 2.3. We will then review the existing literature surveys in Section 2.4 and finally present our survey of state of the art methods for visualizing sequencing data in Section 2.5.

2.1 Biological background

We will start with a brief overview of the biological background and terminology only to the extend that we think is beneficial in better comprehension of the rest of this chapter.

2.1.1 Genome

Each cell of an organism contains a copy of its genome. The genome consists of several long double-stranded DNA bases packaged into compact structures called chromosomes by wrapping around proteins known as histones.

Genes are regions of a genome that contribute to specific functions or traits. Each chromosome may contain many genes. There are an estimated twenty to thirty thousand total genes identified on the human genome with their sizes varying from a few hundred DNA bases to a few million bases. During the gene expression process, segments of a gene’s DNA sequence are replicated into a single stranded molecule called messenger RNA (mRNA). The mRNA is then used as a template to synthesize proteins. These two steps are called transcription and translation respectively and form the central dogma of molecular biology: “DNA makes RNA and RNA makes protein”.

4
2.1.2 DNA sequencing technology

DNA sequencing is the process of determining the order of the base letters (A, C, G, T)’s in DNA. The sequencing starts from the biology lab where DNA molecules are extracted from the cells and are prepared into samples to be processed by a DNA sequencer machine. During the preparation process, the DNA molecules are broken down into small fragments (100’s to 1000’s bases long) through a chemical or physical process. In some studies, special preparation steps are performed (e.g. pulldown) to get a higher concentration of DNA fragments that have specific properties of interest. These fragments are then processed by a DNA sequencer machine and are sequenced separately into digital sequences called sequencing reads or just reads. In Section 2.2.1 we will discuss the different tasks and data flows for processing and analysis of the sequencing reads.

2.1.3 Reference genome

A reference genome is a complete (as much as possible) digital sequence representation of the genome for a specific species. The reference genome of a species does not necessarily represent a specific individual’s actual genome, but is rather a consensus of several individuals of that species. It is pieced together from numerous sequencing reads through a process called sequence assembly (more details in Section 2.2.1). Furthermore, due to technical limitations such as sequencing errors and properties of genome sequences such as repeating regions, reference genomes are almost always incomplete, and are constantly being improved by different organizations, usually international genome consortiums, resulting in the publication of different versions every so often.

2.1.4 Epigenetics

Mechanisms other than changes in the underlying DNA sequence can affect the structural and functional properties of the genome including the expression of the genes. These inheritable changes are broadly referred to as epigenetic changes, where “epi” indicates a change “above” the genome. Examples of such mechanisms are illustrated with the purple tags in the illustrated chromosome in Fig. 2.1 and include chemical modifications to the DNA (DNA methylation), or to its associated proteins (histone modifications). Techniques such as bisulfite sequencing (BS-Seq) and chromatin immunoprecipitation coupled with sequencing (ChIP-seq) can be used to profile and measure the abundance of epigenetic changes across the genome.

In the next section we will discuss more details about the workflow, data types and tasks involved in analyzing sequencing data.
2.2 Sequencing data analysis

Fig. 2.1 demonstrates a high level workflow of sequencing data analysis to the extent that is the scope of this thesis. In this section we will discuss different aspects of this workflow, starting with the data types and the tasks.

2.2.1 Data types and tasks

The sequencing analysis process starts by extracting and preparing DNA (or RNA) from the biological material of interest. These long pieces of DNA are then sheared into millions of small fragments and then put into a DNA sequencer. The sequencing may run for several days depending on the type of the sequencer and the output comes out as millions of short sequences called sequencing reads, raw reads, or simply reads, as illustrated in Fig. 2.1(d1).

These short reads then need to be pieced back together to create the bigger picture of the input DNA sequence. If an existing reference genome of the sequenced species does not exist already, a de novo genome assembly has to be performed to assemble the sequences as illustrated in Fig. 2.1(d2). Optimized genome assembly algorithms can find the overlaps between the millions of short sequences to merge them into a fewer number of longer sequences called contigs, whose size can range from hundreds of thousands to a few million bases each. These contigs then need to be grouped into even longer sequences called scaffolds. However, due to ambiguities caused by different factors such as sequencing errors or long repetitive elements, the assembly of contigs need to be inspected and resolved manually. As such, assembly visualization methods can greatly aid with this process.
The final result of the genome assembly process is a reference genome sequence, illustrated in Fig. 2.1(d3). A reference genome is further studied and compared against other sequencing datasets or other genomes, to locate and annotate the functionally important regions, such as the genes. The information about these regions such as their locations and orientations (strands) are stored in sequence feature data as illustrated in Fig. 2.1(d4).

Most sequencing projects in recent years are from biological samples for which an annotated reference genome has already been created. For those projects, sequence aligner tools are used to map the raw sequencing reads to a reference genome. The result is a collection of a few to hundreds of millions aligned reads, illustrated in Fig. 2.1(d5), each with information such as the mapped locations and the quality of alignment. RNA-Seq is a popular sequencing based technology used to measure expression levels of large numbers of genes. Following sequencing and alignment of reads to a reference genome, the abundance of reads mapped to different genomic regions determines the level of expression for the genes.

Aligned reads can be summarized in different ways. For instance, the number of reads that are mapped to each location (or region) of genome can be counted to create a continuous-valued data, as illustrated in Fig. 2.1(d6). One of the popular uses of the continuous-valued data is in epigenetics. Computational tools called peak-callers [4] can process the continuous-valued data and identify the areas of the genome with epigenetic modifications. The continuous-valued data are also suitable to be used in data-mining and machine learning based methods. A popular method called ChromHMM [5] uses multiple binned and binarized continuous-value datasets as high dimensional input vectors to characterize the regions of the genome into different states.

Sequence variations (polymorphism) can be identified by comparing the base letters of the aligned read sequences to a reference genome. These variations can be small such as changes to single bases (SNPs), illustrated in Fig. 2.1(d7), or insertion or deletion of a few bases (indels), or they can be large such as duplication, deletion or translocation of large (thousands of bases) parts of the genome, called structural variations.

In order to have a high level and global overview of the data, a common aggregation method is to count the number of reads or sum up the continuous-valued data within specific regions of interest. For instance, for gene expression analysis, the reads from multiple RNA-seq experiments are counted per gene to get a measure of the expression level for each gene. These values are then collected into a read count table, such as the one illustrated in Fig. 2.1(d8). A typical read count table may have 6 to 10 columns (3 to 5 replicates per experimental group), and 20K to 30K rows (one row per gene). A common type of analysis, differential expression analysis, performs normalization and statistical analysis on the read count tables to discover significant changes in expression levels between experimental groups.

Recent sample preparation techniques have allowed researchers to gain new kinds of insights utilizing the same core technologies for DNA sequencing. In single cell analysis, during the sequencing pipeline, the fragments from each cell of a biological sample, are
tagged by a random sequence (of 8 to 12 bases), called *unique molecular identifier* (UMI). All reads from a cell get the same UMI barcode, so they can later be separately counted per cell to get a separate read count per gene and cell. These read count values are then stored into a *molecular count matrix* as illustrated in Fig. 2.1(d9). This gives an estimation of the number of molecules that have been observed for each gene in each measured cell. This enables biologists to simultaneously measure the expression of genes for thousands of cells at the individual cell level, providing unique insights into cellular processes in development and diversity of cells. A molecular count matrix is fairly large, with several thousand columns (one column per observed UMI/cell), and 20K-30K rows (one row per gene). However since most of the elements (typically over 95%) are 0, the matrix can be stored efficiently in compressed data formats suitable for large sparse matrices.

Another type of recent techniques are *chromosome conformation capture (3C)* [6], also called 3D genome analysis techniques, which are designed to analyze the spatial 3D organization of chromatin in a cell. In sequencing based 3C methods, such as Hi-C [6] and ChIA-PET [7], the DNA is first cross-linked while still in the chromosome structure, so the parts of DNA which are spatially near each other, are “glued” together before the DNA gets fragmented. As a result, after the DNA is fragmented and sequenced, each sequenced read will map to two regions of the DNA and associate them together. This association is called topologically associating domain (TAD) and is stored in a 2D symmetric matrix structure, called a matrix of *interaction frequencies* or *contact matrix*, as illustrated in Fig. 2.1(d10). A contact matrix may have different or multiple (hierarchical) resolutions, determined by the binning size(s) (i.e. the size of regions represented by each cell) which typically range from a few thousand bases (high-resolution) to a few million bases (low-resolution).

### 2.2.2 EDAM ontology

EDAM [8] is an ontology of concepts in bioinformatics and computational biology. It aims to provide consistent description for the application domain, function, type of input and output data, and available formats of the data to better aid researchers in finding, comparing, selecting and integrating tools into their workflows. We have tried to follow the EDAM ontology and terminology to the best of our abilities and where applicable. However, EDAM is fairly new and under development, so there were some concepts such as single-cell and 3D genome analysis for which we could not find proper EDAM concepts at the time of writing this text.

EDAM includes *topics, operations, data* and *formats* where:

- *topic* is a category denoting a rather broad domain or field of interest, of study, application, work, data, or technology (e.g. “sequence analysis”).

- *operation* is a function that processes a set of inputs and results in a set of outputs (e.g. “alignment”).
• *data* is the type of biological data (e.g. “RNA sequence alignment”).

• *format* is a specific data serialization format (e.g. “BAM”).

EDAM can be downloaded as a single file in the OWL (Web Ontology Language) format\(^1\) or graphically browsed online at several places including the NCBO ontology browser (BioPortal)\(^2\), the EBI Ontology Look-up Service (OLS)\(^3\), and the IFB EDAM Browser\(^4\). For example, Fig. 2.2 shows an example visualization of the EDAM ontology concepts for some sequence alignment file formats.

![Figure 2.2: Visualization of the EDAM ontology concepts for some alignment formats; generated using the EBI Ontology Look-up Service (http://bit.ly/edam_vis_bam)](image)

### 2.2.3 File formats

A recurring challenge when analyzing sequencing data is that quite often different file formats are supported by different tools in different contexts. Despite that, formats that represent similar data types can usually be converted from one to another. As such, rather than focusing on specific file formats, we will be mostly referring to data types when discussing tools or visualization methods. However, we think it would be still beneficial to have a brief overview of the popular file formats used for different sequencing data types.

\(^1\)http://edamontology.org

\(^2\)http://bioportal.bioontology.org/ontologies/EDAM

\(^3\)http://www.ebi.ac.uk/ols/ontologies/edam

\(^4\)https://ifb-elixirfr.github.io/edam-browser
Table 2.1: Sequencing data types and file formats.

<table>
<thead>
<tr>
<th>Data type</th>
<th>File formats (extensions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d1) raw sequencing reads</td>
<td>FASTQ, SRA, SFF</td>
</tr>
<tr>
<td>(d2) assembled sequence</td>
<td>AGP, ACE</td>
</tr>
<tr>
<td>(d3) full genome sequence</td>
<td>FASTA, CFASTA</td>
</tr>
<tr>
<td>(d4) sequencing features (genomic intervals, annotations)</td>
<td>GFF, GTF, BED, bigBED</td>
</tr>
<tr>
<td>(d5) aligned sequencing reads</td>
<td>SAM, BAM, CRAM</td>
</tr>
<tr>
<td>(d6) continuous-valued data (numeric data)</td>
<td>WIG, bigWIG, bedGraph, BED</td>
</tr>
<tr>
<td>(d7) sequence variants</td>
<td>VCF, BCF, GVF, VEP, MAF, GDS, BED</td>
</tr>
<tr>
<td>(d8) count tables</td>
<td>CSV, TSV, XLS</td>
</tr>
<tr>
<td>(d9) sparse matrix of counts</td>
<td>compressed (hdf5), R objects (rds, rdata), CSV, TSV</td>
</tr>
<tr>
<td>(d10) contact matrix (matrix of interaction frequencies)</td>
<td>full matrix, coordinated list, binary (butlr, btr, hic, cool, mcool, h5, homer, HicPro, ginteractions)</td>
</tr>
</tbody>
</table>

Table 2.1 shows the file formats for the sequencing data types explained in Section 2.2.1. Note that some file formats can be used for multiple purposes. The BED file format, for instance, can be used for sequencing features, continuous-valued data, or sequence variants. Also, the data formats for some data types are a super set of others. For example the file formats for aligned sequence reads, may also be used to store unaligned raw sequencing reads, specially when it results in better compressed files. The awesome-bioinformatics-formats website [9] provides an extensive curated list of bioinformatics format specifications and publications. The following is a brief description of the information stored in the file formats for each data type. The (d#) indices refer to the data type indices in Fig. 2.1 and Table 2.1:

- (d1) raw sequencing reads: stores information about each read including the sequence letter and quality score per base, and a unique identifier.

- (d2) assembled sequence: stores information about each contig including number of bases, number of reads and a unique name.

- (d3) full genome sequence: stores the base letters for the full sequence together with an index file specifying the chromosome name and lengths.
• (d4) sequencing features (genomic intervals, annotations): stores information about each feature, including the feature name, the chromosome, start and end coordinates, strand and optional meta information and attributes about features.

• (d5) aligned sequencing reads: stores information about each alignment including the sequence, the chromosome, coordinates, and alignment quality.

• (d6) continuous-valued data (numerical tracks): specify numerical values per genomic location or range.

• (d7) sequence variants: stores information about each genomic variation including the position and alternative sequences.

• (d8) count table: stores the gene expression values computed from sequencing datasets, usually in delimited text formats.

• (d9) sparse matrix of counts: stores the gene expression values per cell, usually in a binary compressed format suitable for sparse matrices.

• (d10) contact matrix (matrix of interaction frequencies): stores the contact (interaction frequency) between pairs of genomic regions in an upper triangular matrix, usually in a compressed binary format with random access.

2.3 A primer on sequencing data visualization

There is a great body of research on methods and software to visualize sequencing data and we will do our best to cover them in Section 2.4 and Section 2.5. But before doing so, we believe a discussion on some common visualization topics will aid on better understanding the sequencing data types and tasks as well as opportunities and challenges in analyzing them using visualization methods.

2.3.1 Genomic coordinates

Genomic locations (loci) and ranges are represented by genomic coordinates. Genomic coordinates are usually specified by a chromosome name, a start position and an end position, on a specific reference genome build. Chromosomes are typically ordered and named by the number of their DNA base-pairs. For example a human genome consists of 23 pairs of chromosomes, with 22 pairs named 1 to 22, plus the X chromosome (a pair in females, one in males) and one Y chromosome (in males only). The genome length is calculated as the sum of the number of DNA bases from one chromosome of each of the chromosome pairs. For example the length of the human and the mouse genomes are about 3.2 billion and 2.5 billion respectively.
When analyzing the sequencing data, it is very important to pay attention to the coordinates used by the data files. Several versions of assemblies exist for most reference genomes and data with coordinates on one genome assembly build will not produce correct results if analyzed together with data with coordinates on another build. Computational tools called liftover tools are used to convert genome coordinates between genome assemblies [10, 11]. Moreover, some file formats such as BED and BAM use 0-based coordinates, i.e. start numbering the nucleotides of a chromosome from 0, while some others like VCF and GFF use 1-based coordinates.

### 2.3.2 Genomic layouts

To create genomic visualizations, the genomic coordinates as well as visual elements have to be mapped to some axis. Nusrat et. al. [12] provide an excellent abstraction of different layouts, abstractions, partitions and arrangement of sequence axes. Here, we will just briefly mention the four most common layouts: linear, circular, space-filling curves and spacial.

![Figure 2.3: Examples of four common genomic layouts: (a) linear layout used in the IGV Web App [13]. (b) full genome mutations visualized in a circular layout using the Circos [14] tool. (c) four iterations of the Hilbert curve. (d) chromatin states compressed along the chromosome using a Hilbert curve. (e) rendering of two chromosome structures in a 3D layout in the HiC-3DViewer [15]. (source: b is from Pleasance et al. [16]; c and d are from Kharchenko et al. [17])](image)

Examples of these four layouts are shown in Fig. 2.3. Fig. 2.3(a) shows an example of a linear layout which is the most common of the four used in most visualizations. It displays an ideogram of the human chromosome 18 on the top and the gene locations underneath, in the IGV Web App [13]. Fig. 2.3(b) is an example of a circular layout used by the Circos [14] tool in which all chromosome ideograms have been placed around the outer ring and information about mutations are displayed densely in the inner rings. Fig. 2.3(c)
illustrates the first four iterations of the Hilbert curve and Fig. 2.3(d) shows the color coded chromatin states compressed along a chromosome [17] using a Hilbert curve. Fig. 2.3(e) shows two chromosomes (17 and 18) in a 3D layout visualized in the HiC-3DViewer [15].

### 2.3.3 Genome browsers

Genome browsers provide a graphical interface to browse, search, retrieve and analyze sequencing data [18]. Data are displayed in horizontal rows called *tracks* with each track representing a data sample such as a sequence feature, a gene expression data, or an epigenetic modification data. The tracks are typically stacked beneath the genome coordinates to facilitate viewing correlations between different data types. The active view can be specified by entering the genome coordinates, using the navigation user interface to zoom and pan, or by searching for features, such as gene names.
Fig. 2.4 shows an example screenshot of tracks visualized in two genome browsers IGV [19] (a-h) and WashU Epigenome Browser [20] (i-l). The chromosome ideogram in Fig. 2.4(a) represents the currently selected chromosome. The user can drag over it to specify the portion of the chromosome to be displayed (indicated by the small red bar). The ruler in Fig. 2.4(b) reflects the visible portion of the chromosome. The tick marks indicate chromosome locations and the span lists the number of bases currently displayed. Note that the 3 parts of the figure are captured separately with each ruler set to different genomic locations and zoom levels.

The density track in Fig. 2.4(c) shows the computed coverage for the aligned reads in Fig. 2.4(d) and indicates the number of reads that are aligned to each base of the genome. Here the genome browser is zoomed in to the base level, so the coverage track displays the depth of the reads displayed at each locus. At each location, if the percentage of the bases of aligned reads that are different from the reference sequence base is larger than a user specified threshold, the colored bars show the proportions of the reads with each base letter.

The alignment track in Fig. 2.4(d) visualizes the sequencing reads mapped to different positions on the reference genome. The reads are stacked when they overlap. The ordering can be specified from various options such as start location, strand, mapping quality, etc. The sequence track in Fig. 2.4(e) shows the reference genome sequence bases for the coordinate range selected in the browser view.

Continuous-valued data, such as epigenetic modifications, are typically visualized by density tracks as shown in Fig. 2.4(f) and (i), or by heatmap tracks as shown in Fig. 2.4(j). The tracks in Fig. 2.4(g) and (h) show two feature tracks. The feature track in Fig. 2.4(g) displays sequencing features created computationally by calling the peaks on the continuous-valued data displayed above it (Fig. 2.4(f)). The feature track in Fig. 2.4(h) is an annotation track showing the genes and their structure.

The three-dimensional conformation of chromosome represented by a contact matrix is usually visualized by either a heatmap, shown in Fig. 2.4(k), or by arcs, as shown in Fig. 2.4(l) [21]. A heatmap is perhaps the most straightforward visualization method for a contact matrix where the colors correspond to the contact counts. The contact matrices are by definition symmetric and the number of rows and columns is equal to the length of the genome divided by the bin size. The arcs visualize the contacts using edges where two locations are connected when the corresponding value in the contact matrix exceeds a user-specified cutoff threshold.

2.3.4 Visual analysis frameworks

In addition to the visualization tools developed for particular application domains, there are tools that are more general purpose and provide a wide range of data management, analysis and visualization capabilities. In different contexts and based on their additional
features, these tools are referred to as frameworks, workflow tools, or workbenches. They are also sometimes called meta tools if they provide an extensible API to create more tools.

**General visual analysis frameworks**

Harger et al. [22] provide a comparison of open source visual analysis toolkits as well as packages for visualization, graph analysis, and statistics. They examine three areas of functionality for each toolkit: visualization functionality, analysis functionality and a description of the development environment(s). With respect to visualization they look at visualizing graphs, trees, tabular and spatio-temporal data. With respect to analysis functionality they look at analysis on graphs and statistical analysis on arbitrary data. Finally, with respect to development environments, they look at several factors including platforms, language bindings, file formats and documentation.

Zhang et al. [23] provide a comparative review of a selection of state-of-the-art commercial visual analysis frameworks as a complementary to the previous survey on open source tools [22]. They give recommendations to potential users on which tools are applicable for what types of applications and identify several improvement opportunities as future research directions. Among the ten systems they surveyed, they reported Tableau\(^5\), Spotfire\(^6\) and Advizor\(^7\) to be the leaders in interactive visualization and automatic analysis. All three systems have roots back in academic research [24, 25] and seem to be putting noticeable effort in integrating innovative visualization techniques.

**Visual analysis frameworks for sequencing data**

We will discuss a few popular visual analysis frameworks for sequencing data, and provide further information later in Section 2.5.

Galaxy [26, 27] is one of the well-known tools that provides support for pipeline workflows mostly aimed for users with lack of expertise in bioinformatics and programming. It also provides a limited set of visualization options through a web-based extension called Trackester [28] and a community driven repository called Galaxy Toolshed [29]. Fig. 2.5(a) shows an example workflow created in Galaxy to generate read count tables from RNA-seq raw reads.

Orange [30, 31] is a general-purpose machine learning and data mining suite for data analysis. It uses Python scripting and visual programming and provides interactive data analysis and component-based assembly of data mining procedures. Orange allows extensions through modules called widgets. Several extensions have been developed for differ-

\(^5\)https://www.tableau.com

\(^6\)https://spotfire.tibco.com

\(^7\)https://www.advizorsolutions.com
ent types of genomics analysis including gene expression, annotation and most recently a stand-alone single cell analysis software bundle, scOrange [32]. Fig. 2.5(b) shows a single cell analysis workflow created in scOrange.

Knime4Bio [33] is a visual modular workflow environment used to analyze sequencing data and is built within the KNIME [34] framework. It incorporates steps from data preprocessing to statistical analysis and visualization of omics scale data.

Last but not least, bioKepler [35] is a suite built in the Kepler [36] scientific workflow system to facilitate development and distributed execution of bioinformatics workflows. It offers a graphical user interface to aid in creating workflows but it does not provide visual analysis capabilities.

2.4 Previous surveys on genomic data visualization

During our literature survey we came across a surprisingly large number of existing papers and resources that collected and reviewed genomic data visualization tools. We have grouped them into four categories: literature surveys aimed at users, literature surveys on specific tasks, online resources, and literature surveys aimed at tool developers.

2.4.1 Literature surveys aimed at users

A large number of literature surveys on genomic data visualization focus on the features of the tools and the analysis tasks supported by them. As such, these surveys are mainly targeted toward users in the biology and bioinformatics community.

Schroeder et al. [37] review effective and common visualization techniques, tools, and resources for exploring and visualizing multidimensional cancer genomics (oncogenomics) data. They discuss tools and techniques in different categories including genome browsers, heatmaps, circular plots, and network visualization and describe four case studies that illustrate the use of those tools for the visual exploration of different cancer data sets.
Pavlopoulos et al. [38] review a large number of visualization and analysis tools, provide a brief summary of each, and group them into several categories including genome browsers, visualization tools for genome alignments, visualization tools for assemblies, visualization tools for comparative genomics, and RNA-Seq analysis viewers. They also present a timeline of the emergence of relevant technologies, trends and concepts for visualization in genomics.

More recently, Qu et al. [39] review methods used in visual analysis of cancer genomics data, “including traditional approaches such as scatter plots, heatmaps, and networks, as well as emerging technologies using AI and VR”. They also demonstrate the development of genomic data visualization tools over time and analyze their evolution.

2.4.2 Literature surveys on specific tasks

As opposed to the surveys in the previous section which cover tools used in a broad range of analysis tasks, there are also surveys that focus on more specific category of tools and tasks.

**Genome browsers**

Wang et al. [18] provide an introduction to web-based genome browsers and give an overview of their main functions and features covering data visualization, retrieval, analysis and customization. They group and discuss genome browsers based on whether they are species-specific or multiple species and provide details on the user interface and main functions of a selected number of them.

Karnik and Meissner [40] provide a guide to epigenome data resources (e.g. histone modifications and DNA methylation data) and review the genome browsers as tools for visualization, hypothesizing, and validation. They provide examples to highlight the key features and demonstrate the utility of the (epi)genome browsers in different scenarios.

**Sequence alignment**

Procter et al. [41] describe the major features and capabilities of a selection of stand-alone and web-based software for visualizing multiple sequence alignment, phylogeny, and gene family evolution. They review software ranging from simple viewers, to systems that provide sophisticated editing and analysis functions. They also provide a discussion of the challenges that these tools face due to the flood of next generation sequence data and the increasingly complex network of bioinformatics information sources.

**Variant analysis**

Pabinger et al. [42] review the tools for three essential analysis steps in variant analysis of sequencing data: variant identification, variant annotation, and visualization where they look at web-based and stand-alone genome browsers and circular plots. More recently, Zhang
et al. [43] provide a survey and evaluation of web-based tools and databases for variant analysis including visualization tools.

**Gene expression analysis**

Chowdhury et al. [44] overview taxonomy and tools for RNA-seq data analysis and provide a survey of tools available for different stages including quality control, normalization, differential expression analysis, and visualization. They also provide a comparison of RNA-seq data visualization tools and discuss visualization at different levels of data flow e.g. at read level, read alignment level, and normalized or unnormalized read count level.

**Epigenetics**

Wei et al. [45] survey web-based tools and databases for different types of computational epigenetics analysis including bisulfite sequencing, histone modification, methylation analysis, and integrative epigenomics and compare their key features as well as input and output formats and types.

**3D genome structure**

Ay and Noble [46] review computational tools to interpret Hi-C data and discuss the entire analysis pipeline all the way from mapping, filtering, and normalization to confidence estimation and visualization.

Goodstadt and Marti-Renom [47] review challenges for visualizing three-dimensional data in genomic browsers. They state the challenges raised by the attributes of multiscale, multistate, time dependence, and uncertainty in 3d genome data and define a taxonomy of 3D tasks outlining essential features of 3D genome visualization. They also explore the challenges and considerations in data representation (grammar, abstraction, and variation), data refining (classification, comparison, and annotation) and data interaction (navigation, selection, and curation).

Yardımcı and Noble [21] review five software tools for visualizing three-dimensional configuration of DNA through Hi-C data. They summarize different aspects of those tools including the visualization methods they use, their data handling and how well do they scale. They also highlight which tools and techniques are best equipped for which specific tasks specifically for visualization of large scale features, i.e. whole genomes and chromosomes, as opposed to local features such as particular regions of interest.

**2.4.3 Online resources**

Numerous bioinformatics tools and libraries are actively developed by researchers and developers and as a result most printed surveys become out-of-date over time. In addition to new software and methods being released, the existing ones can be updated with new
features, or get discontinued and inaccessible. As such, dedicated online directories that provide curated lists and reviews of bioinformatics software can be a useful complementary resource. However one of the essential requirements for these resources is to provide means of keeping their lists of the software and their status up to date or otherwise they will suffer from the same drawbacks as their printed counterparts.

Open community-driven discussion platforms such as Biostars\textsuperscript{8} \cite{biostars} and SEQAnswers\textsuperscript{9} \cite{seqanswers} provide a large resource of topics related to analysis of sequencing data including tools and tutorials. However, due to the forum-based nature of these platforms, the information about the tools is mostly scattered with little structure and usually focused on most popular tools.

The bio.tools\textsuperscript{10} \cite{bio_tools, bio_tools2} portal is an open source registry of software tools and data resources for the life sciences, supported by the European infrastructure for biological information (ELIXIR). Tools are annotated with topics, operations, input and output data types, and supported formats from the EDAM ontology wherever possible. It mostly relies on community support and allows individuals to register new tools upon signing-up for a free account. As of August 2019, the registry includes about 13,000 resources, with depositions from about 1000 individual registrations.

The Bio-TDS (Bioscience Query Tool Discovery Systems)\textsuperscript{11} \cite{biotds} allows end-users to browse the tools in several categories including a visualization category, or retrieve tools of interest by either keyword-based queries and/or free text based questions. The authors also used a set of 229 queries created from the user postings of the Biostars \cite{biostars} discussion platform to evaluate their system and to compare it with other similar systems such as bio.tools.

The omicX platform, formerly called OMICtools \cite{omic_tools} is an online directory that provides an overview of tools in many domains related to genomics, transcriptomics, proteomics and metabolomics. It is a commercial service which requires a paid subscription to access most of the features. The tools are manually curated and classified into different categories including a category on “genome data visualization software tools”\textsuperscript{12}.

MyBioSoftware \cite{mybiosoftware} is a blog-like web page that offers an extensive list of software tools in bioinformatics which also include visualization tools among others. The tools are classified in different categories of analysis tasks such as alignment, and genome and RNA analysis, and a keyword search feature is provided. For each tool, a description, screenshot,
requirements, download links, and citation information are provided. A registration is not required, but the website is ad supported. New entries can be added using an email address and are announced through a twitter feed\textsuperscript{13}.

The 4D Nucleome Network portal \cite{55} has compiled a list of tools available for data analysis and/or visualization of 3D genome structure datasets. The EpiGenie website \cite{56} is another website with a list of epigenetic tools for statistical data analysis and visualization categorized by their corresponding analysis topic. Last but not least, the web-portal for international human epigenome consortium (IHEC) provides a list of tools, with the option to filter by search terms that includes visualization\textsuperscript{14}.

**Awesome lists**

Community-based collections and curated lists on GitHub, categorically referred to as “awesome lists”, are another useful resource to find most recent tools and publications. The “Awesome Biological Visualizations” \cite{57} is a list of web-based interactive biological visualizations. The “awesome-expression-browser” \cite{58} is a list of software and resources for exploring, browsing and visualizing gene expression data. The “awesome-single-cell” \cite{59} provides a curated list of software packages for single-cell data analysis which also offers a section on interactive visualization and analysis. “HiC_tools” \cite{60} is a collection of tools and papers related to Hi-C data analysis which has a section dedicated to visualization.

To keep the lists up to date, these projects encourage contributions and suggestions through submission of pull requests. They also use automated tests\textsuperscript{15} to regularly check for validity of the URLs in their lists as well as the pull requests.

**Repositories provided by frameworks**

Another collection of visualization tools are those developed by frameworks that also provide a repository for hosting and sharing the tools. The Galaxy project\cite{26} allows creating web interface to bioinformatics software including visualization tools to make them accessible to biologists. A version controlled application store like repository, known as the ToolShed \cite{61}, is provided to host and share the Galaxy utilities. Shiny \cite{3} is an open source R \cite{1} package that provides a web framework for building web applications without requiring knowledge of web development frameworks. Shiny is specially popular within the computational biology community due to its native R environment and the easy to use self-service platform called shinyapps.io\textsuperscript{16} for sharing applications on the web. BioJS\cite{62, 63} is another open source

\textsuperscript{13}https://twitter.com/biosoftcn
\textsuperscript{14}http://ihec-epigenomes.org/research/tools
\textsuperscript{15}https://www.rubydoc.info/gems/awesome_bot
\textsuperscript{16}https://www.shinyapps.io
software framework to develop web-based components and visualization tools for different types of biological data and analysis. In addition to providing a JavaScript framework for bioinformatics developers, it provides a centralized registry of components which currently has over 150 components available for reutilization by the bioinformatics community.

2.4.4 Literature surveys aimed at tool developers

As opposed to the surveys in the previous sections which focused on the features and biological tasks, there are surveys which put more focus on the abstractions of data and tasks and visualization techniques, and as a result have more impact for the tool development in the visualization and bioinformatics community.

Nielsen et al. [64] focus on three core user tasks: analyzing sequences in the context of de-novo assembly and in resequencing experiments, browsing annotations and experimental data mapped to a reference genome, and comparing sequences from different organisms or individuals. For each core task, they review several stand-alone and web-based tools and compare their cost, operating systems, and compatibility. The authors also recommend ways of improving the design of these visualization tools, for example by providing a high-level overview of data, or recommendations for where to look at. For genome browsers, they recommend improvements on easy cross-platform access, customization of data and display, and the ability to perform on-the-fly computation within the visualization. In addition, the authors point out the need to improve the integration among tools and ease the transition from one analysis to another.

Kerren et al. [65] reviewed visualization research on biosciences and created BioVis Explorer\(^\text{17}\), an interactive web application for exploration of visualization techniques for biological datasets. It allows investigation of about 150 visualization methods published from 2000 to 2016 by filtering based on types of data (table, network, hierarchy, string), properties of data (temporal, ordinal, spatial, nominal, quantitative, and uncertainty), and tasks (explore, search, filter, select, compare, cluster, annotate, share, and guide). The thumbnails are shown in a two-dimensional layout produced by dimensionality reduction of the distance matrix computed from a weighted sum of the categories, authors, and the year for each publication. New entries can be proposed through a web form and are manually checked before being made publicly accessible.

O’Donoghue et al. [66] summarize key data visualization principles and resources that can help addressing the challenges raised by the increase in volume and complexity of biomedical data. They surveyed the use and challenges of visualization in several biomedical research areas, including genetics, epigenetics, RNA biology, the protein structure, and so on. In each area, the authors discuss emerging frontiers such as three-dimensional genomics and single-cell RNA sequencing. The authors also outline initiatives aimed at improving

\(^{17}\text{http://biovis.lnu.se}\)
visualization practices in biomedical research via better tools, peer-to-peer learning, and interdisciplinary collaboration with computer scientists, science communicators, and graphic designers.

Cruz et al. [67] review visualization models and layouts prominently used in the representation of biological data such as heatmaps, parallel coordinates and hierarchical models, and discuss guidelines for managing visual complexity, for example through clustering. The authors focus on visualizations that use coordinated multiple views to represent high-dimensional multivariate data, in particular time series gene expression, protein-protein interaction networks and biological pathways. They also discuss how those methods can be used to help solve the challenges surrounding the visualization of complex biological data sets.

Recently, Nusrat et al. [12] propose taxonomies for data, visualization, and tasks involved in genomic data visualization and provide a comprehensive review of published genomic visualization tools in the context of those proposed taxonomies. Their proposed data taxonomy describes different encodings for genomic data types as sparse or contiguous feature sets that can be optionally connected within or in between features. Feature sets can be encoded as stacking or overlaying tracks and placed on the coordinate systems with axes that have different configurations for layout (e.g. circular or linear), partition, abstraction and arrangement (e.g. serial or parallel). Visualizations can consist of one or multiple views, each containing a set of aligned tracks and can show data on one or multiple scales and foci. For the task taxonomy they adapt the task typology of Brehmer and Munzner [68] to genomic visualizations and explain how different search and query tasks operate on genomic visualizations. For example in the search tasks *lookup* refers to viewing features at known position, *locate* refers to finding features with desired properties, *browse* refers to reviewing positions with known position and unknown feature set, and *explore* refers to the case where neither the position nor the feature set are known, therefore multiple feature sets at different position in multiple loci are explored.

### 2.5 Our survey on sequencing data visualization tools

Even though we intend to avoid redundancy and replicating the efforts of existing surveys and resources (Fig. 2.6), we still found it beneficial to present a brief survey of visualization tools in the context of the analysis tasks and data discussed in Section 2.2.1.

#### 2.5.1 Collection process

We started our search by general Google scholar search engine and PubMed Central (PMC) as well as several databases including Nature Methods, Nature Biotechnology, Genome Biology, Genome Research, BMC Genomics, Bioinformatics and IEEE. The initial search terms included 'genomic visualization', 'sequencing visualization' and 'genome visual analysis'
Figure 2.6: How surveys proliferate (adapted from xkcd.com/927 and drawn using cmx.io).

considering only papers published after year 2000. The selection process was after manual review and we ended up keeping 215 tools that met the following criteria:

- a software (standalone, web-based, or library) was developed.
- had an essential visualization component, either exploratory or explanatory. So for instance pipeline tools with only file I/O are not included.
- only those libraries that provided direct functions to generate visual output and did not require the user to call other plotting libraries (e.g. ggplot).
- took at least one of the data types in Fig. 2.1 as input. As such, our collection did not include the following analysis: gene regulatory pathways, interaction networks, gene ontology, proteomics, population genetics, quantitative trait locus (QTL), phylogenetics, genome engineering (e.g. CRISPR), secondary structure.
- none-commercial software (full or partial) with either a peer-reviewed or open access publication, or academic citations.
- either the tool itself or some screenshots of the tool were still available for evaluation.

2.5.2 Tools

We will overview a few examples of visualizations for each analysis topic in Section 2.2.1 and provide a list of the related tools. For each tool, we provide a summary of the platforms, features, data types, plot types and most up to date links. To avoid excessive repetition, we
have listed the genome browsers in a separate table and have excluded them from all other lists.

**Quality control of sequencing reads**

Sequencing technologies are not perfect and the acquired data (Fig. 2.1(d1)) may have issues such as artifacts, contamination and low-quality sequenced reads. These errors can originate in different stages from the preparations in laboratory, the sequencer machine, or post processing, and may lead to erroneous conclusions if not addressed properly. Quality control tools such as those listed in Table 2.2, aim to provide summaries to help detecting these issues before further analysis.

Table 2.2: Visualization tools for quality control of sequencing data

<table>
<thead>
<tr>
<th>Title</th>
<th>Platforms</th>
<th>Features</th>
<th>Data types</th>
<th>Plot types</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSeQC [69]</td>
<td>command line</td>
<td>RNA-Seq</td>
<td>aligned reads, sequence, sequence features</td>
<td>boxplot, lines, pie</td>
<td>[website: source]</td>
</tr>
<tr>
<td>NGS QC Toolkit [70]</td>
<td>command line</td>
<td>raw reads</td>
<td>bars, lines, pie</td>
<td>[website: source]</td>
<td></td>
</tr>
<tr>
<td>MultiQC [71]</td>
<td>command line, Python</td>
<td>aligned reads, contact matrix, continuous-valued data, raw reads, sequence variations</td>
<td>bars, lines</td>
<td>[website: source]</td>
<td></td>
</tr>
<tr>
<td>SolexaQA [74]</td>
<td>command line, R</td>
<td>raw reads</td>
<td>bars, heatmap, lines</td>
<td>[website: source]</td>
<td></td>
</tr>
<tr>
<td>FQC Dashboard [73]</td>
<td>command line, web</td>
<td>sequence visualization</td>
<td>raw reads, area, heatmap, lines</td>
<td>[website: source]</td>
<td></td>
</tr>
<tr>
<td>Fastx-toolkit [74]</td>
<td>command line, web</td>
<td>raw reads</td>
<td>bars, heatmap</td>
<td>[website: source]</td>
<td></td>
</tr>
<tr>
<td>FastQC [75]</td>
<td>standalone</td>
<td>sequence visualization</td>
<td>aligned reads, raw reads, sequence variations</td>
<td>boxplot</td>
<td>[source]</td>
</tr>
<tr>
<td>HTQC [76]</td>
<td>standalone</td>
<td>raw reads</td>
<td>bars</td>
<td>[website: source]</td>
<td></td>
</tr>
<tr>
<td>PRINSEQ [77]</td>
<td>standalone, web</td>
<td>dimensionality reduction, sequence visualization</td>
<td>raw reads</td>
<td>bars, boxplot, points, sequence logo</td>
<td>[website: source]</td>
</tr>
</tbody>
</table>

Figure 2.7: Quality control of raw sequencing reads: (a) box plots showing range of quality values across all bases at each position of the raw sequencing reads. (b) read length distribution. (c) frequency plot showing per base sequence content. Figures from FastQC [75] and PRINSEQ [77] documentations.

Fig. 2.7 shows example visualizations for quality control of raw sequencing reads. Fig. 2.7(a) is an output of the FastQC [75] tool and shows an overview of the range of quality values (y-axis) across all bases (x-axis) at each position of the raw sequencing reads. For each base
position the box plot represents the median and different quartile ranges (10%-90% and 25%-75%) and the blue line represents the mean quality where higher score indicates better base call. The read length distribution and the frequency plots in Fig. 2.7(b) and (c) are two of the outputs of PRINSEQ [77] tool and can be used as other measures of the quality of the sequencing runs. For the read length distribution, a normal distribution is expected. The mean length (M) and the length for one and two standard deviations (1SD and 2SD) are also marked to help to decide where to set length thresholds for the data preprocessing.

The frequency plots use sequence logos [78] to show a per base sequence content. Typically an even distribution of all base letters is expected (right side of Fig. 2.7(c)). Otherwise, it could indicate artifacts, such as untrimmed tags attached to the end of sequences, as seen on the left part of Fig. 2.7(c).

### Sequence assembly visualization

Reconstructing the full genome sequence from the short sequencing reads Fig. 2.1(d2) is usually a challenging task due to ambiguities caused by factors such as repetitive elements, sequence errors, and complexity of the underlying data. A list of the tools for assembly visualization designed to aid in identifying and correcting these ambiguities is shown in Table 2.3. Fig. 2.8 shows example outputs of two assembly visualization tools.

#### Table 2.3: Tools for assembly visualization.

<table>
<thead>
<tr>
<th>Title</th>
<th>Platforms</th>
<th>Features</th>
<th>Data Types</th>
<th>Plot Types</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM [79]</td>
<td>command line</td>
<td>sequence</td>
<td>alignment track, lines</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>Abyss-Explorer [80]</td>
<td>standalone</td>
<td>user study</td>
<td>sequence</td>
<td>shapes, graph</td>
<td>[website] (source)</td>
</tr>
<tr>
<td>Gap5 [81]</td>
<td>standalone</td>
<td>raw reads, aligned reads</td>
<td>lines, alignment track</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>EagleView [82]</td>
<td>standalone</td>
<td>variant analysis</td>
<td>sequence</td>
<td>alignment track</td>
<td>[website]</td>
</tr>
<tr>
<td>Hawkeye and AMOS [83]</td>
<td>standalone</td>
<td>multiple sequence alignment</td>
<td>aligned reads</td>
<td>alignment track, bars, lines, treemap [website] (source)</td>
<td></td>
</tr>
<tr>
<td>Tablet [84]</td>
<td>standalone</td>
<td>alignment visualization, sequence visualization</td>
<td>raw reads, aligned reads, sequence features, sequence bars</td>
<td>alignment track</td>
<td>[website] (source)</td>
</tr>
<tr>
<td>Unipro UGENE [85]</td>
<td>standalone</td>
<td>alignment visualization, meta tool, API</td>
<td>raw reads, aligned reads, sequence variations, sequence features, sequence tree, bars, circular layout</td>
<td>[website] (source)</td>
<td></td>
</tr>
</tbody>
</table>

The **scaffold view** of the Hawkeye [83] tool in Fig. 2.8(a) provides a visual interface to navigate the assembly hierarchy and detect mis-assemblies. Sequences are colored based on how much they deviate from the mean, their correlation and whether they are compress or expanded. The **graph panel** of the Abyss-Explorer [80] tool shown in Fig. 2.8 shows the connectivity between contigs in different possible assemblies. The contigs are shaped and colored according to their orientation and how well do they align in the constructed reference genome.

### Motif visualization

Sequence motifs are sequencing letter patterns that may appear in different regions of a genome, or as conserved elements in multiple sequence alignments (Fig. 2.1(d2)). They are
Figure 2.8: Examples of sequence assembly visualization: (a) visual interface to navigate the assembly hierarchy and detect mis-assemblies in Hawkeye [83]. (b) the graph of connectivity between contigs in different possible assemblies visualized by Abyss-Explorer [80].

also in visualizing protein alignments where the number of letters are 20. A list of motif visualization tools, their functionalities and supported data types are shown in Table 2.4.

Table 2.4: Motif visualization tools.

<table>
<thead>
<tr>
<th>title</th>
<th>platforms</th>
<th>features</th>
<th>data types</th>
<th>plot types</th>
<th>links</th>
</tr>
</thead>
<tbody>
<tr>
<td>GimmeMotifs [86]</td>
<td>command line</td>
<td>epigenetics, ChIP-seq</td>
<td>sequence features</td>
<td>sequence logo</td>
<td>[website]</td>
</tr>
<tr>
<td>CodonLogo [87]</td>
<td>command line, Python, Galaxy</td>
<td>sequence, data table</td>
<td>sequence features</td>
<td>sequence logo</td>
<td>[source]</td>
</tr>
<tr>
<td>WebLogo [88]</td>
<td>command line, Python, web</td>
<td>sequence</td>
<td>data table</td>
<td>sequence features</td>
<td>[website]</td>
</tr>
<tr>
<td>ggseqlogo [89]</td>
<td>R</td>
<td>sequence</td>
<td>sequence features</td>
<td>sequence logo</td>
<td>[source]</td>
</tr>
<tr>
<td>motifStack [90]</td>
<td>R</td>
<td>phylogenetic, clustering</td>
<td>sequence</td>
<td>heatmap, sequence logo, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>iceLogos [91]</td>
<td>standalone</td>
<td>sequence</td>
<td>sequence</td>
<td>heatmap, sequence logo</td>
<td>[source]</td>
</tr>
<tr>
<td>iMotifs [92]</td>
<td>standalone</td>
<td>matrix</td>
<td>sequence</td>
<td>sequence logo</td>
<td>[source]</td>
</tr>
<tr>
<td>Sequence Bundles [93]</td>
<td>standalone, web</td>
<td>sequence</td>
<td>bundled lines</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>ALVIS [94]</td>
<td>standalone, web</td>
<td>sequence</td>
<td>bundled lines, points, sequence logo, tree</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>enoLOGOS [95]</td>
<td>web</td>
<td>matrix</td>
<td>sequence</td>
<td>[source]</td>
<td></td>
</tr>
<tr>
<td>pLogo [96]</td>
<td>web</td>
<td>sequence</td>
<td>sequence logo</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>Two Sample Logo [97]</td>
<td>web</td>
<td>sequence</td>
<td>sequence logo</td>
<td>[source]</td>
<td></td>
</tr>
<tr>
<td>MEME SUITE [98]</td>
<td>web</td>
<td>gene ontology</td>
<td>aligned reads, matrix, sequence</td>
<td>sequence logo</td>
<td>[website]</td>
</tr>
<tr>
<td>STAMP [99]</td>
<td>web</td>
<td>aligned reads</td>
<td>sequence logo</td>
<td>[source]</td>
<td></td>
</tr>
<tr>
<td>Seq2Logo [100]</td>
<td>web</td>
<td>multiple sequence alignment</td>
<td>sequence, matrix</td>
<td>sequence logo</td>
<td>[website]</td>
</tr>
</tbody>
</table>

Fig. 2.9(a) shows a sequence logo [78] visualization which is created by stacking letters at each position with each letter’s size proportional to their frequency in the input sequences. It was plotted by the ggseqlogo [89] R package, but a similar plot can be created by any of the tools in Table 2.4 with sequence logo plot option. Fig. 2.9(b) shows a sequence bundle [93] visualization created using the ALVIS [94] tool for the same set of sequences. Each sequence
is represented with a continuous line, and the lines at each position are bundles by the base letter of their corresponding sequence at that position.

**Genome browsers**

Genome browsers are one of the most popular approaches for visualizing genome-scale data and have played an important role in increasing the accessibility to large public data sets. We previously had an overview of the functionality of genome browsers in Section 2.3.3. Table 2.5 provides a list of genome browsers and summary of their features and input types.

**Sequence variant analysis**

Variations in genomes vary in sizes from *single nucleotide variants* (SNVs) illustrated in Fig. 2.1(d7) to structural variations with larger genomic rearrangements that span thousands of nucleotide bases. A list of visualization tools for variant analysis is shown in Table 2.6.

We previously saw example visualizations of variants in genome browsers in Fig. 2.4(c,d). Fig. 2.10(a) shows a rainfall plot created by the GenVisR [138] R library. It is mostly used for detecting hot spots of mutations in cancer genomics, where the x-coordinate shows the genomic position of the mutation and the y-coordinate represents the base pair distance between consecutive mutations on a logarithmic scale [158]. Fig. 2.10(b) shows genomic rearrangements visualized by the Gremlin [151] interactive visualization tool. It consists of three linked views of genome from the complete genome perspective to the specific structural rearrangement view. Rearrangement locations are encoded with an arc and are colored based on the type of rearrangement: deletions, inversions, and inter-chromosomal translocations.
reads from each input data mapped to the genes. It is done using bioinformatics tools such as Gene expression analysis

In gene expression analysis the activity of genes are quantified by counting the number of reads from each input data mapped to the genes. It is done using bioinformatics tools such as
Table 2.6: Visualization tools for variant analysis.

<table>
<thead>
<tr>
<th>Title</th>
<th>Platforms</th>
<th>Features</th>
<th>Data Types</th>
<th>Plot Types</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circos [14]</td>
<td>command line</td>
<td>comparative genomics, structural variation</td>
<td>raw reads</td>
<td>bars, circular layout, ribbons</td>
<td>[website]</td>
</tr>
<tr>
<td>GeneFuse [133]</td>
<td>command line</td>
<td>structural variation</td>
<td>raw reads</td>
<td>alignment track</td>
<td>[source]</td>
</tr>
<tr>
<td>MerSnip [134]</td>
<td>command line</td>
<td>structural variation</td>
<td>raw reads</td>
<td>alignment track</td>
<td>[source]</td>
</tr>
<tr>
<td>CGmapTools [135]</td>
<td>command line</td>
<td>DNA methylation, epigenetics</td>
<td>aligned reads</td>
<td>heatmap, lollipop, tanghulu</td>
<td>[website]</td>
</tr>
<tr>
<td>MetIGo [136]</td>
<td>command line</td>
<td>epigenetics, DNA methylation</td>
<td>aligned reads, sequence variations, sequence features, sequence</td>
<td>bars, bars</td>
<td>[website]</td>
</tr>
<tr>
<td>QuickRNASeq [137]</td>
<td>command line</td>
<td>RNA-Seq</td>
<td>aligned reads, continues-visited data, sequence variations</td>
<td>bars, heatmap, lollipop, rainfall, waterfall</td>
<td>[website]</td>
</tr>
<tr>
<td>RareVariantVis [138]</td>
<td>R</td>
<td>structural variation</td>
<td>aligned reads</td>
<td>bars, points</td>
<td>[website]</td>
</tr>
<tr>
<td>GCmapTools [139]</td>
<td>command line</td>
<td>DNA methylation, epigenetics</td>
<td>aligned reads</td>
<td>heatmap, lollipop, tanghulu</td>
<td>[website]</td>
</tr>
<tr>
<td>MethGo [140]</td>
<td>command line</td>
<td>DNA methylation, epigenetics</td>
<td>aligned reads, sequence variations, sequence features, sequence</td>
<td>bars, bars</td>
<td>[source]</td>
</tr>
<tr>
<td>EagleView [82]</td>
<td>standalone</td>
<td>assembly visualization</td>
<td>sequence</td>
<td>alignment track</td>
<td>[website]</td>
</tr>
<tr>
<td>Bambino [142]</td>
<td>standalone</td>
<td>alignment visualization</td>
<td>aligned reads, sequence</td>
<td>bars, alignment track</td>
<td>[website]</td>
</tr>
<tr>
<td>QuickRNASeq [137]</td>
<td>command line</td>
<td>RNA-Seq</td>
<td>aligned reads, continues-visited data, sequence variations</td>
<td>bars, heatmap, lollipop, rainfall, waterfall</td>
<td>[website]</td>
</tr>
<tr>
<td>RNAseqVis [143]</td>
<td>standalone</td>
<td>sequence visualization</td>
<td>aligned reads</td>
<td>alignment track</td>
<td>[website]</td>
</tr>
<tr>
<td>Meander [144]</td>
<td>standalone</td>
<td>structural variation</td>
<td>continues-visited data, sequence features</td>
<td>bars, bars, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>QualitySNPflag [145]</td>
<td>standalone</td>
<td>haplotypes</td>
<td>aligned reads</td>
<td>bars</td>
<td>[website]</td>
</tr>
<tr>
<td>T-Rex [146]</td>
<td>standalone</td>
<td>structural variation</td>
<td>continues-visited data, sequence features</td>
<td>bars</td>
<td>[website]</td>
</tr>
<tr>
<td>MAQView [147]</td>
<td>standalone</td>
<td>structural variation</td>
<td>continues-visited data, sequence features</td>
<td>bars, alignment track</td>
<td>[website]</td>
</tr>
<tr>
<td>GATK [148]</td>
<td>standalone</td>
<td>structural variation</td>
<td>continues-visited data, sequence features, sequence variations</td>
<td>points</td>
<td>[website]</td>
</tr>
<tr>
<td>RseqFlow [149]</td>
<td>standalone, command line, Python</td>
<td>haplotypes</td>
<td>continues-visited data, sequence features</td>
<td>bars, bars, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>StratomeX [150]</td>
<td>standalone, web</td>
<td>clustering, DNA methylation, filtering</td>
<td>data portal, data table</td>
<td>heatmap, ribbons</td>
<td>[website]</td>
</tr>
<tr>
<td>GeneSig [151]</td>
<td>web</td>
<td>comparative genomics, structural variation, user study</td>
<td>unscalable</td>
<td>bars</td>
<td>[website]</td>
</tr>
<tr>
<td>Variant view [152]</td>
<td>web</td>
<td>structural variation</td>
<td>data table</td>
<td>bars, lollipop</td>
<td>[website]</td>
</tr>
<tr>
<td>MAGI [153]</td>
<td>web</td>
<td>structural variation</td>
<td>data table</td>
<td>heatmap, lollipop</td>
<td>[source]</td>
</tr>
<tr>
<td>UCSC Cancer Genomics Browser [154]</td>
<td>web</td>
<td>structural variation</td>
<td>data portal, sequence variations</td>
<td>heatmap</td>
<td>[website]</td>
</tr>
<tr>
<td>GenoScope [155]</td>
<td>web</td>
<td>structural variation, epigenetics, meta tool</td>
<td>data portal, aligned reads, sequence, data table</td>
<td>heatmap, tree, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>VDJ [156]</td>
<td>web</td>
<td>structural variation, epigenetics, gene expression analysis, RNA-Seq</td>
<td>continues-visited data, data table, sequence variations</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>vcfTable [157]</td>
<td>web</td>
<td>sequence variations</td>
<td>bars</td>
<td>[website]</td>
<td></td>
</tr>
</tbody>
</table>

htseq-count [159] and the values are then collected in the read count tables (Fig. 2.1(d8)). The raw read count tables need to be processed for filtering (e.g. removing low count genes), correction of artifacts (e.g. caused by difference in sample preparation or sequencing technology) and normalization (e.g. to adjust for different sequencing depth). The processed read count tables can then be analyzed by differential expression analysis tools [160, 161] and visualized using tools with support for gene expression analysis. A collection of such tools is shown in Table 2.7.

Fig. 2.11 shows three example visualizations used in gene expression analysis. Fig. 2.11(a) and (b) are outputs of the DEBrowser [167] interactive tool and visualize mean normalized read counts for genes ($\log_{10}$ transformed). Fig. 2.11(a) shows an interactive scatter plot of all genes where the coordinates for each axis are computed from read counts from multiple replicate samples for one of the two biological conditions. The genes are colored based on their differential expression state from one condition to the other: blue when down regulated, red when up regulated and gray when no change detected. Selected genes (e.g. based
on their biological function) are then clustered and visualized in a heatmap in Fig. 2.11(b). Heatmaps encode the values of a data matrix as shades of color and are a popular visual encoding for biological data. The matrix rows are the genes and the columns are the input samples. The matrix values are the normalized read count for the corresponding genes and input samples and are mapped to the color palette specified by the user.

Fig. 2.11(c) shows an output of the StratomeX [150] tool which is designed to provide an interactive interface for exploring large complex and heterogeneous datasets. Each column is a clustering of the patients: on left column shows four groups of patients are clustered by the mRNA expression and on the right column the patients are grouped by their genomic variation status. The bands between these columns show how many patients overlap between them.
Table 2.7: Visualization tools for gene expression analysis.

<table>
<thead>
<tr>
<th>title</th>
<th>platforms</th>
<th>features</th>
<th>data types</th>
<th>plot types</th>
<th>links</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIPER [162]</td>
<td>command line</td>
<td>RNA-Seq</td>
<td>data table, raw reads</td>
<td>bars, heatmap, lines, points</td>
<td>[source]</td>
</tr>
<tr>
<td>PIVOT [163]</td>
<td>R</td>
<td>clustering, dimensionality reduction, RNA-Seq</td>
<td>data table</td>
<td>bars, heatmap, lines, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>TRAPR [164]</td>
<td>R</td>
<td>RNA-Seq</td>
<td>heatmap, lines, points</td>
<td>[source]</td>
<td></td>
</tr>
<tr>
<td>DEeva [165]</td>
<td>R</td>
<td>clustering, dimensionality reduction, RNA-Seq</td>
<td>data table</td>
<td>bars, heatmap, lines, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>Glimma [166]</td>
<td>R</td>
<td>microarray, RNA-Seq</td>
<td>data table</td>
<td>bars, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>DEBrowser [167]</td>
<td>R, Shiny</td>
<td>ChIP-seq, clustering, dimensionality reduction, gene ontology, normalization, quality control, RNA-Seq</td>
<td>data table</td>
<td>bars, heatmap, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>VioR [168]</td>
<td>standalone</td>
<td>API (Java), API (R), clustering, dimensionality reduction, epigenetics, filtering, meta tool, RNA-Seq, single cell</td>
<td>aligned reads, continuous-valued data, data table, molecular count matrix</td>
<td>alignment track, bars, boxplot, circular layout, heatmap, image, lines, points, venn, violin</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>VioExpress [169]</td>
<td>standalone</td>
<td>clustering, API (R), filtering</td>
<td>data table</td>
<td>heatmap, wordcloud, lines, bars</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>Mayday [170]</td>
<td>standalone</td>
<td>microarray, clustering, dimensionality reduction, machine learning, API (R)</td>
<td>database, data table</td>
<td>heatmap, tree, lines, bars</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>Chipster [171]</td>
<td>standalone</td>
<td>ChIP-seq, clustering, DNA methylation, gene ontology, microarray, RNA-Seq</td>
<td>data table</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>UEA eDNA workbench [172]</td>
<td>standalone</td>
<td>RNA secondary structure, RNA-Seq</td>
<td>raw reads</td>
<td>alignment track, boxplot, lines, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>RuBiNA [173]</td>
<td>standalone</td>
<td>meta tool, RNA-Seq</td>
<td>aligned reads, raw reads</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>Gitools [174]</td>
<td>standalone</td>
<td>clustering, gene ontology</td>
<td>data table</td>
<td>heatmap</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>geWorkbench [175]</td>
<td>standalone</td>
<td>meta tool, clustering, machine learning, plugins, API (Java), gene ontology, filtering</td>
<td>sequence, data table</td>
<td>heatmap, 3D, lines, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>Orange [30]</td>
<td>standalone</td>
<td>API (Python), clustering, data mining, dimensionality reduction, filtering, gene ontology, machine learning, meta tool, RNA-Seq</td>
<td>data table, database</td>
<td>bars, boxplot, heatmap, image, lines, points, tree, venn, wordcloud</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>StratomeX [152]</td>
<td>standalone, web</td>
<td>clustering, DNA methylation, filtering, RNA-Seq, variant analysis</td>
<td>data portal, data table</td>
<td>heatmap, ribbons</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>WebMCV [176]</td>
<td>web</td>
<td>clustering, dimensionality reduction, RNA-Seq, variant analysis</td>
<td>data table</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>Wanderer [177]</td>
<td>web</td>
<td>epigenetics, DNA methylation</td>
<td>data portal</td>
<td>lines, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>MEXPRESS [178]</td>
<td>web</td>
<td>DNA methylation, epigenetics, RNA-Seq</td>
<td>data portal</td>
<td>bars, lines</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>MethHC [179]</td>
<td>web</td>
<td>clustering, DNA methylation, epigenetics, microarray, RNA-Seq</td>
<td>data portal</td>
<td>heatmap, heatmap, lines, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>GenePattern [180]</td>
<td>web</td>
<td>API (Java), API (Matlab), API (Python), API (R), clustering, meta tool, RNA-Seq</td>
<td>data table, sequence variations</td>
<td>heatmap, lines, points</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>Expression Profiler [181]</td>
<td>web</td>
<td>clustering, filtering, microarray</td>
<td>data table</td>
<td>heatmap, lines</td>
<td>[source]</td>
</tr>
<tr>
<td>VOE [356]</td>
<td>web</td>
<td>ChIP-seq, epigenetics, RNA-Seq, variant analysis</td>
<td>continuous-valued data, data table, sequence variations</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>DRIVA [182]</td>
<td>web</td>
<td>filtering, RNA-Seq</td>
<td>data table</td>
<td>points</td>
<td>[source]</td>
</tr>
<tr>
<td>MethSurv [183]</td>
<td>web, R</td>
<td>epigenetics, DNA methylation</td>
<td>data portal</td>
<td>heatmap, violin</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>IDEA [184]</td>
<td>web, R, Shiny</td>
<td>data table</td>
<td>[website] [source]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Visualization for epigenetics

Epigenetic modifications are changes to the chromosome structure that affect the functionality of DNA without modifying the underlying sequence. Examples of such mechanisms include chemical modifications to the DNA itself or to its associated proteins and are referred to as “epigenetic marks”. Table 2.8 shows a list of visualization tools for analysis of epigenetic datasets.

We previously saw example visualizations of epigenetic marks in genome browsers in Fig. 2.4(f, i, j). Fig. 2.12(a) shows an example heatmap created from three epigenetic marks using the deepTools2 [187] tool. This type of heatmap configuration is popular
Table 2.8: Visualization tools for epigenetics analysis.

<table>
<thead>
<tr>
<th>Title</th>
<th>Platforms</th>
<th>Features</th>
<th>Data Types</th>
<th>Plot Types</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGinupTools [135]</td>
<td>command line</td>
<td>DNA methylation, variant analysis</td>
<td>aligned reads</td>
<td>heatmap, killprop, tanglevis</td>
<td>[website]</td>
</tr>
<tr>
<td>MethGe [138]</td>
<td>command line</td>
<td>DNA methylation, variant analysis</td>
<td>aligned reads, sequence variations, sequence features</td>
<td>lines, bars</td>
<td>[source]</td>
</tr>
<tr>
<td>ViewB [135]</td>
<td>command line</td>
<td></td>
<td></td>
<td>heatmap</td>
<td>[source]</td>
</tr>
<tr>
<td>GimmicMotive [66]</td>
<td>command line</td>
<td>motif visualization, ChIP-seq</td>
<td>sequence features</td>
<td>[website]</td>
<td>[source]</td>
</tr>
<tr>
<td>ChromBHM [5]</td>
<td>command line</td>
<td>machine learning</td>
<td>sequence features</td>
<td>heatmap</td>
<td>[website]</td>
</tr>
<tr>
<td>snpPlot [186]</td>
<td>command line, R</td>
<td>feature visualization, API (R)</td>
<td>aligned reads, continuous-valued data, sequence features</td>
<td>heatmap, lines</td>
<td>[source]</td>
</tr>
<tr>
<td>deepTools2 [187]</td>
<td>command line, web</td>
<td>clustering, dimensionality reduction, feature visualization, quality control</td>
<td>aligned reads, continuous-valued data, sequence features</td>
<td>heatmap, lines</td>
<td>[website]</td>
</tr>
<tr>
<td>HilbertVis [188]</td>
<td>R</td>
<td>ChIP-seq</td>
<td>continuous-valued data, sequence features</td>
<td>points, hilbert layout</td>
<td>[website]</td>
</tr>
<tr>
<td>HilbertCurve [189]</td>
<td>R</td>
<td></td>
<td></td>
<td>hilbert layout</td>
<td>[source]</td>
</tr>
<tr>
<td>MethVisual [139]</td>
<td>R</td>
<td>DNA methylation, clustering</td>
<td>sequence features</td>
<td>heatmap, killprop, points</td>
<td>[source]</td>
</tr>
<tr>
<td>BeToons [190]</td>
<td>R</td>
<td>DNA methylation, clustering</td>
<td>sequence features, data table</td>
<td>heatmap, lines</td>
<td>[website]</td>
</tr>
<tr>
<td>methylPipe and compEpTole [192]</td>
<td>R</td>
<td>DNA methylation, clustering, gene ontology, ChIP-seq</td>
<td>aligned reads, sequence features</td>
<td>heatmap, tree</td>
<td>[website]</td>
</tr>
<tr>
<td>methylSnit [193]</td>
<td>R</td>
<td>DNA methylation, clustering</td>
<td>aligned reads, continuous-valued data</td>
<td>pie, points, bars</td>
<td>[source]</td>
</tr>
<tr>
<td>genomation [194]</td>
<td>R</td>
<td>ChIP-seq, annotation</td>
<td>aligned reads, continuous-valued data, sequence features</td>
<td>heatmap, lines</td>
<td>[source]</td>
</tr>
<tr>
<td>SeqPlots [195]</td>
<td>R, Shiny, standalone, web</td>
<td>ChIP-seq, clustering, feature visualization, machine learning, RNA-Seq</td>
<td>aligned reads, continuous-valued data, sequence features</td>
<td>heatmap, lines</td>
<td>[source]</td>
</tr>
<tr>
<td>Volt [196]</td>
<td>standalone</td>
<td>API (Java), API (R), clustering, dimensionality reduction, filtering, gene expression analysis, meta tool, RNA-Seq, single cell</td>
<td>aligned reads, continuous-valued data, data table, molecular count matrix</td>
<td>alignment track, bars, boxplot, circular layout, heatmap, image, lines, points, vows, violins</td>
<td>[website]</td>
</tr>
<tr>
<td>Chalk [196]</td>
<td>standalone</td>
<td>feature visualization, DNA methylation, clustering, ChIP-seq, filtering</td>
<td>continuous-valued data, sequence features</td>
<td>heatmap, lines</td>
<td>[website]</td>
</tr>
<tr>
<td>BiQ Analyzer [197]</td>
<td>standalone</td>
<td>DNA methylation</td>
<td></td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>seqMIXER [198]</td>
<td>standalone</td>
<td>ChIP-seq, clustering, RNA-Seq</td>
<td>aligned reads, continuous-valued data</td>
<td>bars, heatmap, lines, points</td>
<td>[source]</td>
</tr>
<tr>
<td>Spark [199]</td>
<td>standalone</td>
<td>clustering, DNA methylation, gene ontology, RNA-Seq</td>
<td>continuous-valued data, sequence features</td>
<td>heatmap, lines, tree</td>
<td>[website]</td>
</tr>
<tr>
<td>Bowtie [200]</td>
<td>web</td>
<td>3D genome, clustering, ReC</td>
<td>aligned reads, continuous-valued data, data portal, database, sequence features</td>
<td>area, bars, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>EpilEplorer [201]</td>
<td>web</td>
<td>DNA methylation</td>
<td>sequence features</td>
<td>workflow, lines, bars</td>
<td>[website]</td>
</tr>
<tr>
<td>3Disease Browser [202]</td>
<td>web</td>
<td>3D genome</td>
<td>data portal</td>
<td>heatmap, vows, lines, points, bars, 3D layout</td>
<td>[website]</td>
</tr>
<tr>
<td>Wanderer [177]</td>
<td>web</td>
<td>DNA methylation, gene expression analysis</td>
<td>data portal</td>
<td>lines, points</td>
<td>[website]</td>
</tr>
<tr>
<td>MEXPRESS [210]</td>
<td>web</td>
<td>DNA methylation, gene expression analysis, RNA-Seq</td>
<td>data portal</td>
<td>bars, lines</td>
<td>[source]</td>
</tr>
<tr>
<td>MethHC [219]</td>
<td>web</td>
<td>clustering, DNA methylation, gene expression analysis, microarray, RNA-Seq</td>
<td>data portal</td>
<td>barplot, heatmap, points</td>
<td>[website]</td>
</tr>
<tr>
<td>CMS [200]</td>
<td>web</td>
<td>DNA methylation</td>
<td>data portal</td>
<td>heatmap</td>
<td>[website]</td>
</tr>
<tr>
<td>Gnomecore [155]</td>
<td>web</td>
<td>structural variation, meta tool, variant analysis</td>
<td>data portal, aligned reads, sequence data table</td>
<td>heatmap, tree, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>VOR [216]</td>
<td>web</td>
<td>ChIP-seq, gene expression analysis, RNA-Seq, variant analysis</td>
<td>continuous-valued data, data table, sequence variations</td>
<td></td>
<td>[source]</td>
</tr>
<tr>
<td>MethTools [204]</td>
<td>web, command line</td>
<td>DNA methylation</td>
<td>sequence features</td>
<td>heatmap, killprop, points</td>
<td>[website]</td>
</tr>
<tr>
<td>RNAtools [205]</td>
<td>web, R</td>
<td>DNA methylation</td>
<td>sequence features</td>
<td>heatmap, violins, points</td>
<td>[website]</td>
</tr>
<tr>
<td>MetStoR [183]</td>
<td>web, R</td>
<td>DNA methylation, gene expression analysis</td>
<td>data portal</td>
<td>heatmap, violins</td>
<td>[website]</td>
</tr>
<tr>
<td>Epitro [206]</td>
<td>web, standalone, R</td>
<td>API (R)</td>
<td></td>
<td></td>
<td>[source]</td>
</tr>
</tbody>
</table>

in epigenomics where each column is a matrix corresponds to a single epigenetic mark (Fig. 2.1(d6)). The rows of each matrix are genomic regions, and the columns are the genomic regions divided to a fixed number of bins. The matrix values correspond to the sum of the continuous-valued data at each bin and are mapped to the color palette specified by the user. The heatmap is also clustered using k-means clustering of the three epigenetic marks with $k = 2$.  

32
ChromHMM [5] is a popular method that uses a multivariate Hidden Markov Model (HMM) on a set of continuous-valued epigenetic marks to annotate the genome into states called chromatin states. A common configuration used for ChromHMM has 15 hidden states. The emission probabilities of these states for 5 major epigenetic marks are visualized as a heatmap in Fig. 2.12(b) where the darker blue color corresponds to a greater probability of observing the mark in the state. Each state is also manually annotated with a color and a label based on the biological characterization of the state.

ChromHMM can also assign each location of the genome to an instance of each of the learned states and export a genome-wide map of the chromatin state annotations. These annotations can then be visualized in a genome browser. Fig. 2.12(c) shows a compact and summarized visualization of 127 different sets of chromatin states learned for 127 cell or tissue types with 5 marks for each (i.e. a total of $127 \times 5$ marks). The visualization is done by the WashU Epigenome Browser [120] on the data generated by the NIH Roadmap Epigenomics Consortium [207]. At the top panel a stacked barplot called epilogos [208] adapts the idea of sequence logo visualization of motifs to the chromatin states at each genomic location. In the middle panel, each row corresponds to the chromatin states learned from the epigenetic marks for one cell or tissue type and the bottom row shows the gene annotations.

Single cell analysis

Single cell RNA-seq (scRNA-seq) enables high-throughput gene expression profiling at the individual-cell levels [209]. The gene expression values per cell are stored in a molecular count matrix (Fig. 2.1(d9)) as explained in Section 2.2.1. Each cell in a single-cell molecular count matrices can be treated as a high dimensional vector where the dimensions are the gene expression values counted for that cell. Many bioinformatics analysis methods are
based on general dimensionality reduction techniques. The overall pipeline consists of filtering (abnormal or low count cells and genes), dimensionality reduction (PCA followed by tSNE), clustering (k-means, hierarchical, density peaks, etc.) and differential expression analysis. Different clustering methods are employed by different tools, such as k-means ([210]), hierarchical ([211, 212]), spectral ([213]), graph-based ([214]), model-based ([215]), grade of membership models ([216]), and density peaks ([217]). They also use different similarity metrics such as euclidean distance ([211, 218, 214]), Pearson correlation ([210, 212]), and multi-kernel learning ([213]). So visualization methods, a list of which is shown in Table 2.9, are invaluable in gaining insights into each of these pipeline steps and verifying their effectiveness.

Table 2.9: Visualization tools for single cell analysis.

<table>
<thead>
<tr>
<th>title</th>
<th>platforms</th>
<th>features</th>
<th>data types</th>
<th>plot types</th>
<th>links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scater</td>
<td>R</td>
<td>clustering, dimensionality reduction</td>
<td>molecular count matrix</td>
<td>heatmap, points</td>
<td>[website]</td>
</tr>
<tr>
<td>Monocle</td>
<td>R</td>
<td>clustering, dimensionality reduction</td>
<td>molecular count matrix</td>
<td>points</td>
<td>[website]</td>
</tr>
<tr>
<td>Seurat</td>
<td>R</td>
<td>clustering, dimensionality reduction</td>
<td>molecular count matrix</td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>CountClust</td>
<td>R</td>
<td>clustering, RNA-Seq</td>
<td>molecular count matrix</td>
<td>bars</td>
<td>[website]</td>
</tr>
<tr>
<td>SC3</td>
<td>R</td>
<td>clustering</td>
<td></td>
<td>heatmap</td>
<td>[website]</td>
</tr>
<tr>
<td>RaceID</td>
<td>R</td>
<td>clustering, dimensionality reduction</td>
<td></td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>CIDER</td>
<td>R</td>
<td>clustering, dimensionality reduction</td>
<td></td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>RCA</td>
<td>R</td>
<td>clustering</td>
<td>molecular count matrix</td>
<td>heatmap, points</td>
<td>[website]</td>
</tr>
<tr>
<td>CellTree</td>
<td>R</td>
<td>clustering, dimensionality reduction</td>
<td></td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>SIMLR</td>
<td>R, MATLAB</td>
<td>clustering, dimensionality reduction</td>
<td>molecular count matrix</td>
<td>heatmap, points</td>
<td>[website]</td>
</tr>
<tr>
<td>FIt-SNE</td>
<td>R, Python, MATLAB</td>
<td>dimensionality reduction</td>
<td>molecular count matrix</td>
<td>heatmap</td>
<td>[website]</td>
</tr>
<tr>
<td>VisR</td>
<td>standalone</td>
<td>clustering, dimensionality reduction, epigenetics, filtering, gene expression analysis, meta tool, RNA-Seq</td>
<td>aligned reads, continuous-valued data, data table, molecular count matrix</td>
<td>alignment track, bars, boxplot, circular layout, heatmap, image, lines, points, venn, violin</td>
<td>[website]</td>
</tr>
<tr>
<td>scOrange</td>
<td>standalone</td>
<td>clustering, dimensionality reduction, gene ontology, filtering, API (Python)</td>
<td>molecular count matrix</td>
<td>boxplot, violin, lines, points, bars</td>
<td>[website]</td>
</tr>
<tr>
<td>TSCAN</td>
<td>web, R, Shiny</td>
<td>clustering</td>
<td>molecular count matrix</td>
<td>heatmap, lines, points, bars</td>
<td>[website]</td>
</tr>
</tbody>
</table>

Figure 2.13: Example visualization of single cell RNA-seq data using the Monocle [217] R package: (a) tSNE projection and density peak clustering (b) projection of cells and coloring using pseudo-time (c) coloring cells by cluster on the pseudo-time projection (c) gene expressions for three genes across the pseudo-time.
Fig. 2.13 shows some of the analysis steps of the Monocle [217] package. Fig. 2.13(a) is a scatter plot of cells in a two-dimensional embedding computed by tSNE dimensionality reduction. The cells are colored by a cluster ID (computed using density peaks) but it is also common to color the cells by the expression levels of the genes of interest. Even though it is expected that pairs of single cells that are similar in the high-dimensional space (i.e. have similar gene expression patterns) will be close together in the low dimensional space, it is important to know the pitfalls and short comings to avoid misinterpretation of the results [221].

Monocle uses a method called *manifold learning* to order the cells in a pseudotime and placing them along a trajectory. Fig. 2.13(b) and (c) shows an example trajectory which has a tree-like structure and corresponds to a biological process such as cell differentiation. The cells in the trajectory can be colored based on their pseudotime (Fig. 2.13(b)) or a state inferred from the segments of the tree (Fig. 2.13(c)). Last but not least, as shown in Fig. 2.13(d), expression levels of selected marker genes can be plotted against the pseudotime to verify that the pseudotime ordering looks good.

**Visualization of 3D genome structure**

The functionality of the genome is affected by the proximity and interaction with the regulatory elements in the 3D structure of the chromosome, referred to as TADs. As stated in Section 2.2 the frequency of these interactions is measured using methods referred to as 3C and are stored as contact matrices (Fig. 2.1(d10)). Table 2.10 shows a list of tools for visualization of 3D genome structure using the contact matrices.

![Figure 2.14: Example visualizations of 3D genomic data: (a) heatmap and 3D representation of the chromatin conformations visualized using the TADkit [224] web-based tool. (b) contact frequencies visualized with arcs in a circular layout using the Rondo [200] web-based tool.](image)

We previously saw example visualizations of contact matrices in genome browsers in Fig. 2.4(k, l). Fig. 2.14 shows two other example visualizations using tools specifically developed for visualization of 3D genomic data. Fig. 2.14(a) shows the interface of TADkit [224]
Table 2.10: Tools for visualization of 3D genome structure

<table>
<thead>
<tr>
<th>title</th>
<th>platforms</th>
<th>features</th>
<th>data types</th>
<th>plot types</th>
<th>links</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiC-inspector</td>
<td>command line</td>
<td>raw reads</td>
<td>heatmap</td>
<td>[source]</td>
<td></td>
</tr>
<tr>
<td>HiCPlotter</td>
<td>command line, Python</td>
<td>continuous-valued data, sequence features, data table</td>
<td>heatmap, arcs, bars, area</td>
<td>[source]</td>
<td></td>
</tr>
<tr>
<td>TADbit</td>
<td>command line, Python, web</td>
<td>Hi-C</td>
<td>contact matrix, raw reads</td>
<td>3D, 3D layout, bars, heatmap</td>
<td>[website] [source]</td>
</tr>
<tr>
<td>HTCA</td>
<td>R</td>
<td></td>
<td>heatmap</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>Juicebox</td>
<td>standalone</td>
<td>Hi-C</td>
<td>heatmap</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>GMOL</td>
<td>standalone</td>
<td>mesh</td>
<td>3D, 3D layout</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>GenomeFlow</td>
<td>standalone</td>
<td>contact matrix, mesh, raw reads, sequence features</td>
<td>heatmap, 3D, 3D layout</td>
<td>[source]</td>
<td></td>
</tr>
<tr>
<td>Galaxy</td>
<td>web</td>
<td>Hi-C</td>
<td>contact matrix</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>Juicebox.js</td>
<td>web</td>
<td>Hi-C</td>
<td>heatmap, bars</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>Higlass</td>
<td>web</td>
<td>Hi-C, API (JavaScript)</td>
<td>heatmap, bars</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>HiPile</td>
<td>web</td>
<td>Hi-C, user study</td>
<td>heatmap</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>HiGlass</td>
<td>web</td>
<td>Hi-C</td>
<td>heatmap</td>
<td>[website] [source]</td>
<td></td>
</tr>
<tr>
<td>Rondo</td>
<td>web</td>
<td>clustering, epigenetics, Hi-C</td>
<td>aligned reads, continuous-valued data, data portal, database, sequence features</td>
<td>arcs, bars, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>HiC-3DViewer</td>
<td>web</td>
<td>Hi-C, ChIP-seq</td>
<td>3D layout</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>3DGBR</td>
<td>web</td>
<td>Hi-C</td>
<td>3D layout</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>My5C</td>
<td>web</td>
<td>sequence features, sequence</td>
<td>heatmap, lines</td>
<td>[website]</td>
<td></td>
</tr>
<tr>
<td>3Disease Browser</td>
<td>web</td>
<td>epigenetics</td>
<td>data portal</td>
<td>heatmap, venn, lines, points, bars, 3D layout</td>
<td>[website]</td>
</tr>
</tbody>
</table>

web-based visualization tool which provides a linked view between a heatmap visualization of the TAD contact matrices and a 3D representation of the chromatin conformations. TADkit is a companion interactive front-end to the TADbit tool which generates the 3D models of selected genomic domains. It uses the Integrative Modeling Platform (IMP) [236] to satisfy the spatial restraints computed from the input contact matrices.

Fig. 2.14(b) shows another visualization of TADs in a circular layout using the Rondo [200] web-based tool. Rondo uses clustering to group the connections to reduce the visual clutter. An interactive interface allows to customize the view and different visual properties.

Visual analysis frameworks

We discussed some of the most popular visual analysis frameworks in Section 2.3.4. A list of visual analysis frameworks with support for sequencing data is shown in Table 2.11. Among the plethora of open-source and commercial data mining and visualization tools available, here we have listed those which are either designed solely for the purpose of genomics analyses or provide some extensions or support for them.
Table 2.11: Visual analysis frameworks for sequencing data.

<table>
<thead>
<tr>
<th>title</th>
<th>platforms</th>
<th>features</th>
<th>data types</th>
<th>plot types</th>
<th>links</th>
</tr>
</thead>
<tbody>
<tr>
<td>VisIR [168]</td>
<td>standalone</td>
<td>API (Java), API (R), clustering, dimensionality reduction, epigenetics, filtering, gene expression analysis, RNA-Seq, single cell</td>
<td>aligned reads, continuous-valued data, data table, molecular count matrix</td>
<td>alignment track, bars, boxplot, circular layout, heatmap, image, lines, points, venn, violin</td>
<td>[website]</td>
</tr>
<tr>
<td>EaSeq [237]</td>
<td>standalone</td>
<td>ChIP-seq</td>
<td></td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>Geneious Basic</td>
<td>standalone</td>
<td></td>
<td></td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>RobiNA [173]</td>
<td>standalone</td>
<td>gene expression analysis, RNA-Seq</td>
<td>aligned reads, raw reads</td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>bioKepler [239]</td>
<td>standalone</td>
<td></td>
<td></td>
<td></td>
<td>[website]</td>
</tr>
<tr>
<td>Unipro UGENE [85]</td>
<td>standalone</td>
<td>alignment visualization, assembly visualization, API (C++)</td>
<td>raw reads, aligned reads, sequence variants, sequence features, sequence</td>
<td>tree, bars, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>geWorkbench [175]</td>
<td>standalone</td>
<td>clustering, machine learning, plugins, API (Java), gene expression analysis, gene ontology, filtering</td>
<td>sequence, data table</td>
<td>heatmap, 3D, lines, points</td>
<td>[website]</td>
</tr>
<tr>
<td>Orange [30]</td>
<td>standalone</td>
<td>API (Python), clustering, data mining, dimensionality reduction, filtering, gene expression analysis, gene ontology, filtering, machine learning, RNA-Seq</td>
<td>data table, database</td>
<td>bars, boxplot, heatmap, image, lines, points, tree, venn, wordcloud</td>
<td>[website]</td>
</tr>
<tr>
<td>Galaxy [29]</td>
<td>web</td>
<td>API (Perl), API (Python), API (R), clustering, filtering, machine learning, microarray, RNA-Seq</td>
<td>aligned reads, continuous-valued data, data portal, data table, database, raw reads, sequence, sequence features, sequence variations</td>
<td>alignment track, area, bars, boxplot, circular layout, graph, heatmap, hilbert layout, image, lines, points, sequence logo, venn, violin</td>
<td>[website]</td>
</tr>
<tr>
<td>GenePattern [180]</td>
<td>web</td>
<td>API (Java), API (Matlab), API (Python), API (R), clustering, gene expression analysis, RNA-Seq</td>
<td>data table, sequence variations</td>
<td>heatmap, lines, points</td>
<td>[website]</td>
</tr>
<tr>
<td>Genohub [155]</td>
<td>web</td>
<td>structural variation, epigenetics, variant analysis</td>
<td>data portal, aligned reads, sequence, data table</td>
<td>heatmap, tree, circular layout</td>
<td>[website]</td>
</tr>
<tr>
<td>knimeEFILE [34]</td>
<td>web, standalone</td>
<td>gene ontology</td>
<td>aligned reads, continuous-valued data, sequence variations, sequence features</td>
<td></td>
<td>[website]</td>
</tr>
</tbody>
</table>
Chapter 3

An interactive analysis and exploration tool for epigenomic data

Our first project in sequencing data visualization was a collaboration with researchers in Canada’s Michael Smith Genome Sciences Centre and BC Cancer Agency on analysis of epigenomics datasets. We followed the design study methodology throughout this project. Sedlmair et al. [240] define a design study as “a project in which visualization researchers analyze a specific real-world problem faced by domain experts, design a visualization system that supports solving this problem, validate the design, and reflect about lessons learned in order to refine visualization design guidelines”.

In this chapter, we present our design study: an analysis and abstraction of the data and tasks related to the domain of epigenomics, and the design and implementation of ChAsE, an interactive tool to facilitate data analysis and visualization in this domain. Using ChAsE epigenomic data can be grouped into subsets either by clustering or by querying for combinations of presence or absence of signal (on/off) in different epigenomic experiments. These steps can be interleaved and the comparison of different workflows is explicitly supported. We took special care to contain the exponential expansion of possible on/off combinations by creating a novel querying interface. An interactive heatmap facilitates the exploration and comparison of different clusters. We validated our iterative design by working closely with two groups of biologists on different biological problems. Both groups quickly found new insight into their data as well as claimed that our tool would save them several hours or days of work over using existing tools.

3.1 Introduction

Most cells in an organism share the same underlying DNA sequence (genome) and yet they display a great diversity of physical properties and functions. This diversity largely comes from differences in which genes are active (expressed) or silent (repressed) in each cell type.
As discussed in Chapter 2, these changes in gene expression caused by mechanisms other than changes in the underlying DNA sequence are broadly referred to as epigenetic changes, examples of which include chemical modifications to the DNA itself or to its associated proteins. We will refer to these chemical modifications as “epigenetic marks” throughout this chapter.

Techniques such as ChIP (chromatin immunoprecipitation) coupled with innovative DNA sequencing technology (ChIP-seq) allow measuring the abundance of epigenetic marks across the genome, giving rise to so called “epigenomic data”. Many large consortia such as ENCODE [241] and the NIH Epigenomics Roadmap Project [242] have convened to exploit these technologies and perform hundreds of ChIP-seq experiments involving diverse cell types. The key challenge for new biological insight lies in integrative analysis, in which different data are combined and interpreted together, for example as patterns of epigenetic marks across different cell types.

While computational methods to interpret these data continue to evolve and improve, there is great value in data exploration and many questions remain too ill-defined to be addressed in an automated fashion. Visualization is thus a valuable tool in this domain. In addition, the rapidly changing computational tool set for data analysis often requires significant computational expertise to use. Many of the biologists who possess the detailed knowledge needed to interpret these data must rely on programming experts. As a result, interactive visualization holds great promise in being able to lower the computational barrier to analysis and engage biology experts more directly in data processing and interpretation.

In this chapter, we address this need for visual analysis tools and present an interactive tool for visual exploration and analysis of epigenomic data. Our first contribution is a characterization of the data (see Section 3.2) and a discussion and abstraction of the related domain tasks (see Section 3.3). Second, we provide our design including an interactive heatmap explorer and approaches for querying combinations and subsets (see Section 3.5). Third, we validate our approach by presenting two detailed case studies with two groups of domain experts working on different biological problems, and reporting their insights (see Section 3.6). We also comment on lessons learned in our design process and provide suggestions for other researchers working in this domain.

3.2 Biological background and data

3.2.1 Epigenetic marks

Due to noise introduced at various stages of the ChIP-seq procedure, the resulting measurements of epigenetic marks are not binary values corresponding to the presence/absence of a given chemical modification at each position in the genome. Rather, ChIP-seq provides measurements of epigenetic mark enrichment across the genome and filtering methods [4, 243] are used to distinguish signal “peaks” from background noise. Each peak has a chromo-
some start and an end position and enrichment values across the interval. These peaks are usually stored as continuous-valued data (see Section 2.2.1) most commonly the Wiggle format [244, 245] and can be browsed as tracks in genome browsers. Researchers may refer to these data as marks, wig files, track data, samples, or experiments, but we refer to them as “epigenetic marks” or simply “marks” throughout this chapter.

### 3.2.2 Region sets

In order to make sense of epigenetic marks across the genome, researchers often focus on genomic regions defined by features of biological interest. One common example is the set of start positions of known genes referred to as transcription start sites (TSS). A region set is a collection of genomic intervals and is usually described in sequence feature data formats (see Section 2.2.1) such as the GFF (General Feature Format) [246] or the BED (Browser Extensible Data) [247] file formats. These formats capture the genomic locations of the regions and support inclusion of additional information, such as external database identifiers for each feature. The number of regions within the region set depends on the type of the analysis, but it usually varies between a few hundred to tens of thousands.

Often, the genomic intervals are of a fixed length centred on features of interest, for example, ±1,000 nucleotides (nt) around a TSS. It is also quite possible for them to be of different lengths, for example, the boundaries of annotated genes. Our users all asked for a customizable fixed length interval and requested that variable length regions either be extended or truncated to this fixed length.

### 3.2.3 Data abstraction

Many analysis tasks involve investigation of multiple epigenetic marks across a single region set. This allows us to consider only the subset of epigenetic mark values that fall within the target regions. Since the epigenetic marks and the region set use a common reference genome coordinate system, the mapping is straightforward. The result is a set of high-dimensional vectors, where each vector contains the values of a given epigenetic mark across a single region. If we define $m$ as the number of epigenetic marks under consideration, $r$ as the number of regions in the set, and $l$ as the region length (same for all regions in the set), then this process will produce $m \times r$ vectors of length $l$.

In order to perform computationally efficient analysis on these vectors and to reduce artifacts such as signal spikes, it is common practice to accumulate multiple positions of each vector into a single bin. Our users typically used bin sizes of 50 to a few hundred, which have biological meaning in terms of nucleotide length. So, for example, regions of size 2,000 nt and a bin size of 50 nt, will result in vectors containing only 40 values. The binned values are normalized with methods such as the sigmoid function used by ChromaSig [248] to enable comparison between multiple epigenetic marks.
3.3 Task analysis

Data peaks are a very common starting substrate for analysis (see Section 3.2.1) and there are a great number of different questions biologists might wish to ask that can help reveal the functional role of the epigenetic marks. Common ones include “are there genes nearby and what are they?” or “do these peaks lie in regions with characteristic patterns?”.

Through our discussions with several biologists, we identified several tasks that were not well served by existing tools. While solutions existed, they either did not immediately produce an interactive visual output to allow exploration and verification of the results, or did not capture all the requested functionality. Here we categorize these tasks and highlight important considerations gleaned from discussion with analysts.

3.3.1 Task 1 - signal query

Considerations: (1) It remains open for debate whether the signal height from an epigenetic mark has biological significance. Given that they are acquired through an enrichment process as discussed in Section 2.2.1, height is important for distinguishing signal from noise; however it is unclear whether more subtle height differences are meaningful. Biologists therefore tend to reason about peaks as being either present or absent. (2) An individual epigenetic mark is almost never considered in isolation. Part of the power of the ChIP-seq technology is to profile the positions of several modifications in the same cell or tissue type in parallel and integrate the results into a meaningful picture of the larger system. (3) While many analyses are exploratory, biologists very often have a particular signal pattern in mind. A common workflow is to query for regions with a pattern of interest and then explore from this starting point.

Example Question: “Show me the regions where there are peaks in marks A and B, but not in C or D. I don’t care about the peak status in E through G.”

3.3.2 Task 2 - cluster

Considerations: (1) Clustering is a powerful exploration tool and is best used in cases where no precise query can be formulated. (2) It is also used in categorizing differences in signal position or distribution. (3) Researcher often want to explore the output of Task 1 through clustering.

Example Question: “In my target region set, what are the classes of data patterns in marks E through G?”

3.3.3 Task 3 - quality control

Considerations: (1) k-means clustering is widespread and well-known in the biology community. It has the well-known drawback that it requires the number of clusters as input (which requires informed guess work). k-means is also using random seeds, creating different
clusters each time it is run. (2) Datasets “in the wild”, including sequencing data, often do not have an obvious cluster structure, i.e. clusters often overlap. (3) Biologists desire to visually inspect the clusters to either assure the reliability of the clusters, try a different cluster number or do further downstream analysis of the found clusters.

**Example Question:** “Do most regions in the first cluster follow the trend of signal presence in mark A and absence in mark C?”

### 3.3.4 Task 4 - comparison

**Considerations:** (1) Biologists very often want to find the intersection of sets. For example, clusters of interest could be obtained through iterations of Task 1 and 2. A biologist may then want to compare the clusters from different workflows to determine whether they contain the same or different regions. (2) Generation of many different intersections can be laborious and comparison of the output in visualization modules is cumbersome.

**Example Question:** “Do subset 1 and subset 2 contain the same or different regions?”

### 3.3.5 Task 5 - downstream analysis

**Considerations:** Biologists will frequently need to generate visual and text outputs of their results and findings either to (1) read them into other tools for further analysis, (2) to communicate them with their peers or (3) to include them in manuscripts.

**Example:** “I want to use another tool to check the functional similarities of the regions in this subset.”

### 3.4 Related work

As discussed earlier in Section 2.3.3, genome browsers are a popular approach for visualizing genome-scale data and play an important role in increasing the accessibility of large public data sets, such as the ENCODE data resource currently hosted by the UCSC Genome Browser [249]. Each epigenetic mark is displayed as a separate heatmap or histogram plot called a “track”, and then multiple marks can be viewed simultaneously by vertically stacking these tracks. Part of the power of this arrangement is that data from diverse marks are anchored to the same horizontal reference coordinate and can thus be readily compared.

Genome browsers are optimized for viewing one local region at a time. While this makes them valuable for detailed data inspection and exploration, it prevents them from aiding in global pattern analysis. Several techniques have emerged to facilitate global pattern discovery in epigenomic data. These include probabilistic methods for the discovery of epigenetic signatures de novo, such as ChromaSig [248], Hidden Markov Model approaches, specifically ChromHMM [5] and Bayesian network approaches [250] to uncover recurrent epigenetic states. However, these methods require significant computational skill to use and in most cases remain inaccessible to most biologists.
There are a handful of applications that attempt to bridge this computational gap in epigenomic data analysis. For example, CisGenome [113] contains a graphical interface for running analyses such as peak detection, false discovery rate computation and sequence analysis. Similarly, seqMINER [198] offers a range of data processing capabilities including an implementation of $k$-means clustering and a corresponding heatmap display. This type of clustering and heatmap view have been widely accepted for epigenomic data ever since their appearance in early analysis papers [251, 252]. Cistrome [253] provides integrative analysis and visualization tools for ChIP-seq data, taking advantage of the Galaxy platform [26]. The strength of these tools lies in their ability to connect diverse analysis methods in a single application. While they provide visualization components, the emphasis is on chaining tools into a workflow rather than on optimizing the visual representation and there is very little linking between the different visual displays.

Spark [199] provides a visual workflow to address clustering and while it does a good job in helping to explore different clusters (Task 2), it doesn’t provide a way to understand the variance of the clusters (Task 3) nor does it allow the user to query on/off combinations (Task 1) nor does it facilitate comparisons (Task 4).

Understanding combinatorial combinations (Task 1) has been hard and does not scale well. Practical implementations are typically constrained to just very few sets and are often visualized using Venn- or Euler-diagrams [254]. Alternative representations of combinatorial queries use iconic representations or a Karnaugh map [255].

StratomeX [150] provides visual subset comparison using ribbons of varying width drawn between neighboring columns. While this visual encoding works for comparison of multiple different cluster results, we specifically focus on just comparing two different results here. While we can envision to integrate some of its functionality into a future version of our tool, our focus was the integration of Tasks 1-5 in one simple tool at this point.

Finally, standard visual and interaction concepts such as Brushing and Linking [256] as well as Dynamic Queries [257] are commonplace in today’s visualization tools such that they are well understood by our users. Hence, our tool is making extensive use of these concepts.

### 3.5 ChAsE

We now describe our tool, called ChAsE (Chromatin Analysis and Exploration), and outline how our current approach addresses the analysis tasks discussed in Section 3.3.

#### 3.5.1 Data input

A graphical user interface shown in Fig. 3.1 allows specifying one or more epigenetic marks and one region set of genomic intervals. Processing parameters, such as the normalization options, or visualization parameters, such as heatmap colour or ordering, can be specified
per epigenetic mark. A visibility option was added after we observed that users preferred to load a larger set of epigenetic marks upfront and then modify it depending on their immediate analysis goals. The region size and number of bins need to be specified only once as they will be identical for preprocessing all epigenetic marks.

The processing time depends on the size of the input files but usually takes a few minutes per data file. The results of the processing are stored in the output directory specified by the user, so future data loading times will be much faster (a few seconds). Users can reopen the input dialog during analysis and modify the input parameters or add or remove marks without losing the current state of their analysis.

### 3.5.2 Interface

The ChAsE interface consists of five linked panes as shown in Fig. 3.2. Data from a single region set and one or more epigenetic marks is first loaded into the Workspace Pane (Fig. 3.2(a)). We will refer to this as the “full set”. It can then be divided into various subsets, which we will simply call “set” or “sets”, using functionality within the three alternate Method Panes (Fig. 3.2(b), Fig. 3.4). Once created, a set can be stored in the Favourites Pane (Fig. 3.2(e)) for later use. The plots can be inspected in a zoomed view in the Plot Pane (Fig. 3.2(d)). Closer inspection of data across individual regions is reserved for the Heat Map Pane (Fig. 3.2(c)).
3.5.3 Workspace Pane

The Workspace Pane (Fig. 3.2(a)) shows a snapshot of the current sets and is organized as a matrix. Each row of this matrix corresponds to one set and a column corresponds to a particular epigenetic mark. We chose a data representation commonly used by biologists in the field, called a “profile plot”. The x-axis captures offsets from the region start (e.g. position relative to a TSS) and the y-axis is used to express a summary statistic for all values at these relative positions. A profile plot summarizes the data for each epigenetic mark in each set (i.e. for each cell in the matrix) providing the user with a quick visual summary of the data patterns.

Offset from the main matrix, the leftmost column displays a summary of all the epigenetic marks in one row as overlaid profile plots which we call a “summary plot”. Comparing signal distributions across many columns can be challenging and the summary plot offers a valuable mechanism for spotting subtle differences in signal distributions between epigenetic marks. To aid this comparison, the summary plot and profile plots are linked, such that when a user mouses over a column, its corresponding curve in the summary plot is highlighted. The size of each set is shown in square brackets as a percentage of the full set or the actual number of regions when the size drops below 1%. Each set can also have a user specified title or descriptive note allowing the users to keep track of their history. Clicking on either the summary plot or any individual profile plot automatically displays it in the
Plot Pane (Fig. 3.2(d), lower right) and the Heat Map Pane (Fig. 3.2(c), upper right) for closer inspection.

Profile plot views

A user can alternate between different profile plot visualizations and their size through a context menu. We provide four choices of summary statistics for display in the profile plots (Fig. 3.3): The Mean and standard deviation view (3.3a) shows the average signal profile of the region set surrounded by the +/- standard deviation range. The Continuous box plot view (3.3b) shows the median signal surrounded by the quartile boundaries as an indication of the range and frequency of signal heights. The Mean and signal scatter view (3.3c) shows the average signal profile as well as a scatter of all profiles for the region set accumulated and rendered with a log scale. The Mean and peak scatter view (3.3d) accumulates only the max peak value per region for each epigenetic mark rather than the entire profile, addressing users’ expressed interest in the distribution of the peaks in a set in terms of their height and location.

Figure 3.3: Profile plot views: (a) mean +/- standard deviation, (b) continuous box plot, (c) mean and signal scatter, (d) mean and peak scatter, and (e) summary plot.

3.5.4 Method panes

There are three Method Panes (Fig. 3.4) to address Tasks 1, 2, and 4 outlined in Section 3.3: Signal Query, Cluster, and Comparison. Only one of these three Method Panes is displayed at a time and they always appear at the top of the Workspace Pane. We found, this minimized confusion and allowed the user to focus on a single method.

Signal Query Pane

We explored several different data encodings and interaction schemes to help the user specify a particular signal query. Displaying all possible on/off (i.e. present/absent) combinations across all epigenetic marks would quickly lead to an overwhelming number of options and was impractical. So, it was important to enable our users to limit the combinations by specifying whether a signal should be on or off or either in each epigenetic mark.

We initially tried providing constraints that allowed the user to express multiple combinations at once. For example, a user could specify both the on and off state for mark A and just the on state for mark B. This would give rise to three sets: A-on and B-on, A-off and B-on, and the rest. However, this approach of expressing combinations was not
intuitive to our users. Instead, during our discussions, they would often simply draw out the combinations of interest, one at a time. The number of combinations our users wished to generate tended to be small compared to the space of possibilities. The process could be thought of as querying for individual signal sets and we therefore decided to support this one-at-a-time querying more directly.

Fig. 3.4a shows our final Signal Query Pane design. A user opens this pane by selecting a target set and choosing "Signal Query" from the top “Methods” menu. A pair of check boxes appears above each column and allows the user to specify on (top box checked), or off (bottom box checked), or indifference (neither checked). As the user modifies the query, a preview of the resulting set is shown at the bottom of the pane as a row of profile plots as well as on the heatmap to the right of the pane. In cases where the resulting set is empty no plot or heatmap will be shown. The resulting set is only imported into the Workspace once the user clicks the "Add" button. This allows the user to accumulate sets in their Workspace Pane when there are multiple desired combinations, as well as to make the sets available for further analysis (i.e. clustering or comparison). Annotations are shown above the created
sets in the workspace showing the query used to create the subset (an example shown at the bottom of the Workspace Pane in Fig. 3.2(a)).

Cluster Pane

A user initiates clustering by selecting the target set and selecting “$k$-means clustering” from the “Methods” menu. In addition to the number of clusters, the Cluster Pane allows specifying the epigenetic marks to be included in the clustering step using the check boxes above each mark (Fig. 3.4b). Clicking the 'Run' button commences the clustering run. Once the process is complete, the resulting clusters appear in the Workspace Pane as the children of the input set in a tree structure. The heatmap view shows the clusters separated by horizontal lines with a thicker stroke used to indicate the marks included in the clustering.

Because clusters can be subsequently subclustered, we needed to manage potentially large tree structures. Leaf nodes (i.e. clusters with no sub-clusters) are represented with solid circles, whereas parent nodes (i.e. clusters with sub-clusters) are represented by either a $\subseteq$ sign, to indicate an expanded node, or with a $\supseteq$ sign, to indicate a collapsed node. Allowing the user to toggle between expanded and collapsed states by clicking on the parent nodes made the tree structure manageable.

The user can explore the newly created clusters or choose to rerun $k$-means clustering with the same or different parameters. It is a known fact that the result of $k$-means will depend not just on the value $k$, but also on the initial seeding consisting of $k$ randomly selected members of the input set. Thus each run of $k$-means can result in a different clustering. We kept this as the default behavior to avoid artificially hiding this drawback of the $k$-means algorithm, however we allow the user to specify an integer number to be used as a seed through a menu option. Our users were aware of this fact and tended to run $k$-means with random seed until an interesting clustering is observed or until they could assess the reproducibility of a cluster (part of Task 3).

Comparison Pane

Comparisons can be formulated as queries for the intersection across multiple sets (Task 4). A comparison is initiated by selecting two or more sets from the Workspace Pane while pressing the Shift key and then selecting “Cluster Comparison” from the “Methods” menu. As shown in Fig. 3.4c, the Comparison Pane displays the input sets and a preview of the intersection using the same profile plot display found in the Workspace Pane. The check boxes on the left of the summary plots allows the user to specify either inclusion (checked) or exclusion (unchecked) of the set and a label is shown for clarification of the set operation. Initially all check boxes are checked, thus the result is the intersection of all sets.

48
3.5.5 Heat Map Pane

Heatmaps are one of the most widely used visual encodings for biological data [258]. They encode the values of a data matrix as shades of colour. Heatmaps are used in different stages of the research from data analysis to presentation, but despite their popularity, there are valid arguments against their use [259]. It is much harder to compare signal variations and the overall signal shape from colour variations alone, so other encodings such as profile curves [260, 261], are used alternatively. In addition, the resolution of the data is usually higher than the resolution of the heatmap, and therefore the pixels represent an average and can hide certain data characteristics such as local peaks. We partially address these concerns with our interactive heatmap.

Fig. 3.2(c) depicts the Heat Map Pane. Each column corresponds to a single epigenetic mark and the rows correspond to genomic regions. To render the heatmap, the values of the data matrix are mapped to the colour specified by the user. Users tend to use different colors for marks with different biological nature. We provide a set of six sequential colour schemes with the same perceived intensity as well as two diverging colour schemes which we picked using ColorBrewer [262]. Regions can be sorted by the signal in one mark at a time. All columns are coordinated such that the row order is the same across columns. An arrow above a column indicates the mark currently dictating the sorting. The direction of the arrowhead indicates the sort order and can be flipped when clicked. Underneath the heatmap, a legend provides the total number of the regions, the display density (regions/pixel row), and the current sorting criteria.

Regions are initially sorted by their order in the input regions file, but different sorting criteria, such as signal average, signal max, or signal peak offset, can be chosen by the user through a context menu or the top “Heatmap” menu. Sorting is commonly used to get an overview of the distribution of the signal value and shape across regions of an epigenetic mark, comparing the correlations between multiple epigenetic marks, as well as visually assessing the quality and variations within the clusters (Task 3). Regions belonging to a collapsed parent, will all be sorted together. For an expanded parent, regions belonging to different children will be sorted separately. Fig. 3.5 shows the heatmaps for a single epigenetic mark using different sorting criteria.

Figure 3.5: Heatmaps sorted by: (a) genomic location, (b) signal average, (c) signal maximum (peak), and (d) signal peak offset. (e) profile plots shown on mouseover.
As stated above, heatmaps suffer from at least two major shortcomings. First, detecting the shape of the signal from the colour variation is not straightforward. To address this, as the user drags the mouse pointer on the heatmap we show a profile plot of the regions underneath the heatmap (Fig. 3.5e). Second, when too many regions are overlapped and averaged within one row of pixels, we allow users to interactively zoom and pan through the heatmap. This is realized by a resizable scrollbar on the right side of the heatmap. The scrollbar thumb represents the viewable portion of the entire heatmap. Dragging the up and down arrows on the scroll thumb will resize the Heat Map Pane’s range thus changing the horizontal pixel density. Dragging the solid region of the scrollbar thumb will scroll through the regions.

3.5.6 Plot Pane

The plot pane (Fig. 3.2(d)) shows a zoomed version of the selected profile plot and includes additional legends and axis labels. One of our users showed most interest in a zoomed summary plot where the profiles for all epigenetic marks for a region set are overlaid together. Hence, the user can adjust the horizontal and vertical view range through resizable scrollbars. A small view of the entire plot is shown on the top right corner with the invisible view range shaded.

3.5.7 Favorites Pane

We were often asked by the users to be able to save the current results of their analysis or partial hypothetical findings before performing different tasks. Although it is possible to have all of them in the workspace view and save them to file, this would have cluttered the workspace view in the long run competing with the goal of quick and easy access to the current working sets. We thus provided a favourite pane (Fig. 3.2(e)), where region sets within the workspace could be added for future reference and brought back to the workspace as needed. Regions in the favourite pane are shown with a summary plot and their size.

3.5.8 Common functionality

In addition to the functionality specific to each view specified above, most views share some common functionality, which are available through contextual menus. This functionality includes operations such as annotation, removal, and export of region sets. Further, the user can save images of the heatmap or profile plots as high resolution PDF files (Task 5).

3.6 Evaluation

Our design process had three phases: (1) iterations of interface sketches based on feedback we received from domain experts, (2) implementation of an initial prototype interface based on these refined sketches, and (3) an iterative refinement of the prototype based on feedback
from biologists after using the prototype. During this last phase, we first gave the users a tutorial on the use of the prototype. We then loaded their data and observed them using our tool. During this session, the users offered out loud descriptions of their thoughts while using the tool. We then collected more reflective feedback after deployment of ChAsE for several weeks. Here we present illustrated walkthroughs of two case studies with two groups of collaborators.

3.6.1 Case study 1: signal querying and clustering

Our first group of collaborators were two biologists who were researching the co-localization of patterns across four marks in human liver cells under two conditions C1 and C2.

Analysis 1: filtering using the signal query

Using a set of regions centred on peaks collected from four marks, labelled HNF4a_C1, HNF4a_C2, FoxA2_C1 and FoxA2_C2, our collaborators’ first step was to filter for regions containing signal from two or more of the four marks. This corresponds to 11 out of the possible 16 combinations of presence or absence of signal in four marks. Using the Signal Query Pane, they first identified the four sets in which signal is present for only one of the four marks (3.6(a)). Next, they used the Comparison Pane to exclude these sets from the full set by intersecting their complements (Figure 3.6(b)). While this took less than a minute, a similar workflow with their previous tools would have required them to extract each of the 11 combinations, taking them tens of minutes.

Analysis 2: finding patterns using the signal query

Our collaborators then wanted to identify those regions with signal in HNF4a_C1 and FoxA2_C1, but not in HNF4a_C2 and FoxA2_C2. This would have been frustrating, if not impossible, to achieve with k-means clustering alone, but it was readily performed using the Signal Query Pane (Figure 3.7).

Our collaborators then scanned across the resulting profile plots and inspected the data patterns for seven additional marks not used in the query step. Several observations resulted that confirmed their predictions for the regions in the set created using signal query:

1. H3K4me3_C1 and H3K4me3_C2 are only weakly associated with these regions. This is consistent with previous observations [263, 251]

2. H3K4me1_C1 and H3K4me1_C2 differ in their distributions across the regions; The H3K4me1_C1 mark displays a distinct bimodal (two peaks) distribution, whereas H3K4me1_C2 appears unimodal (single peak). This observation is also consistent with previous reports [263].
Figure 3.6: Analysis 1: Comparison Pane is used to exclude regions in which only one mark out of four showed a signal.

3. These regions have very low levels of H3K27me3_C1 and H3K9me3_C1, which is expected for transcriptionally active sites.

4. H4ac_C1 mimics the bimodal distribution pattern of H3K4me1_C1, also reported previously [251].

Identification of such subsets based on data presence or absence can otherwise be done using an intersection tool, such as that provided in Galaxy [26] or written in custom code. Each intersection must be performed separately and stored for subsequent loading into different tools that provide profile plot or heatmap views. This makes the generation of multiple sets laborious and the comparison of profile plots from other marks difficult. Our tool markedly shortened the time our collaborators needed to generate filtered sets of interest and also provided instant feedback regarding the corresponding data patterns in other marks.
Analysis 3: chaining querying and clustering

A more detailed inspection of the heatmap and brief exploratory sorting of the columns revealed a small set of H3K4me1_C1 with a unimodal rather than a bimodal distribution. Subsequent $k$-means clustering on that mark isolated the unimodal set (highlighted in Figure 3.8). While unimodal profiles for H3K4me1 have been observed previously in other conditions [263], the unimodal pattern for H4ac is undocumented and warrants further investigation.

This analysis illustrated the value of enabling users to interleave their steps of analysis while providing them with visualizations to support quality control in the process.
3.6.2 Case study 2: exploration with the interactive heatmap

Our second group of collaborators were a biologist and a bioinformatician who were studying the relationship between several different marks in mouse embryonic stem cells. For this analysis, the region set consisting of about 30,000 regions in the neighbourhood of characterized genes (TSS +/- 1,000 base pairs) and a total of six different marks were loaded into the tool and labelled CpG, 5-mC, 5-hmC, H3K4me3, H3K27me3, and TET1.

Analysis 1: initial exploration using the heatmap

Unlike in Case Study 1, this group of collaborators wanted to use their original unfiltered data, which was guaranteed to have some low and noisy signals in most regions. This prevented them from taking advantage of the Signal Query or the Comparison Panes effectively and they only employed the clustering and heatmap browsing in their analysis. To further support this task, we introduced a divergent colour scheme to make it easier to judge whether the data values are low, medium or high (blue, yellow, and red, respectively).

Initial browsing of the data in the Heat Map Pane while sorting the regions by the average value of different marks, showed co-localization of CpG and H4K4me3, but an anticorrelation with 5-mC and 5-hmC. This is shown in Figure 3.9 and is consistent with the previous published studies [264].

![Figure 3.9: Heatmap sorted by the average signal of CpG column. This figure is a direct unaltered PDF export from the tool.](image-url)
Analysis 2: coupling clustering with the interactive heatmap

Our collaborators then added a fifth mark H3K27me3 and experimented with different clusterings using the Cluster Pane. Their initial hypothesis was that when 5-hmC and H3K4me3 are present, H3K27me3 should be absent. In biological terms, this would indicate that 5-hmC is present at transcriptionally active genes, where H3K4me3 is high and H3K27me3 is low. By clustering on 5-hmC and H3K4me3 only, our collaborators noticed that the cluster with both 5-hmC and H3K4me3 also unexpectedly showed some H3K27me3 signal (top cluster in Figure 3.10).

![Image](image.png)

Figure 3.10: Clustering by H3K4me3 and 5-hmC into four clusters. A cluster with a high level of H3K4me3 and 5-hmC, but low level of 5-mC is highlighted with a border (top cluster).

This observation led them to explore the pattern using a different clustering based on H3K4me3 and H3K27me3 alone. As shown in Figure 3.11, they were able to identify regions where 5-hmC and H3K4me3 are present and H3K27me3 is absent (top), as originally predicted, but also uncovered another class of regions in which all three are present at high/moderate levels (middle), which enabled them to rule out their original hypothesis. This observation was later confirmed to be consistent with recently published results [265].

To investigate the possible biological reasons for these patterns, our collaborators added a sixth mark, TET1. They observed that the top cluster despite having medium to low values of 5-mC and 5-hmC had a high value of TET1. In biological terms, TET1 is a protein that facilitates a chemical change from 5-mC to 5-hmC. Thus our collaborators were able to conclude that any 5-hmC produced from 5-mC in the presence of TET1 is present only transiently and is presumably rapidly further processed. This was a valuable insight.

The Cluster Pane facilitated these analyses by providing a simple interface to specify the number of clusters and the marks to be included in the clustering. Once the clustering
was complete, a preview of the result was immediately shown as a tree view of profile plots in the Workspace Pane and as sub-clusters sorted individually in the Heat Map Pane. This allowed our users to quickly observe the variation within the clusters and check for existence of interesting patterns, and to rerun the clustering to test if it was stable.

This study showed the value of supporting gradual exploration. In the past, our collaborators had used scripting and Matlab for analysis and similar steps had taken them much more effort to accomplish. This use case provides an example of how clustering is best used for pattern discovery at the point when the researcher wishes to perform an exploratory analysis or wants to isolate a set of regions based on data distribution and not pure presence/absence of signal.

This study also showed the usefulness of the heatmap view to reveal variation and spatial patterns within clusters. For instance, in Figure 3.11 the difference between top and middle clusters is much more visible from the heatmap compared to just the profile plots.

3.7 Lessons learned

Perhaps one of the most difficult aspects to get right was the ability to deal with a quickly expanding set of possible combinations. In several previous iterations of our design, our signal querying pane enabled the user to create combinatorial combinations of on/off behaviours. This often resulted in too many combinations being displayed of which the user was simply interested in a subset. Only after restricting this interface to query exactly one of these combinations at a time did we resolve the usability issues. This was possible after realizing that our users really only needed to analyze a very few and very specific combinations and hence, it was best to have them query them one-by-one. Although some of these tasks could be done in other tools, these tools were sufficiently complicated to use that the effort was not in balance with the payoff.
Further, besides assuring flexible output formats enabling a proper downstream-analysis, all of our users were very keen on functionality that would let them produce high-resolution figures for their publications and communication of their results to peers. There is perhaps a key set of functions that should be provided with most tools, including, but not limited to the export of high-resolution images in standard formats, annotation of features of interest, and customization of colour maps, labels and fonts. It is important to note, that the image resolutions for publications should often be higher than the typical screen resolution.

3.8 Future work

Through discussions with our users, we have identified a number of possible extensions to ChAsE. These include providing improved guidance to the user on the choice of $k$ used during the clustering step, in addition to providing metrics of cluster stability. Being able to reproduce a particular clustering would also be desirable (current clustering is sensitive to initial cluster seeds). There is also potential to provide visual feedback on some of the upstream processing steps such as peak-calling, not addressed here. For example, many peak-calling tools remove peaks with a max height below some threshold. Being able to interactively tune that threshold and visualize the results would be of great value.
Chapter 4

A visual framework for analysis of sequencing data

In Chapter 3 we presented our design study to develop a visual analysis tool to support a common but specific workflow in epigenomics data analysis. During the development of ChAsE, we encountered several instances where we had to spend a great deal of time to implement features which already existed in command-line tools or libraries, but could not be easily integrated into our tool due to platform and architecture limitations. This so-called reinventing the wheel included, for example, reading sequencing data, clustering, and rendering the heatmap and profile plots. There were also secondary requirements which we could not address, for example detecting the peaks from raw continuous-valued data or browsing regions of interest in a genome browser. Even though most of the computational tool sets for further analyses are already available through command line interfaces and libraries, however they often require significant computational expertise to use and many of the biologists with the knowledge needed to interpret these data must rely on programming experts.

In this chapter, we present our solution, a framework called VisR that provides a computationally rich and accessible framework for integrative and interactive analyses of sequencing data without requiring programming expertise. We achieve this aim by providing R-apps, which offer a semi auto-generated and unified graphical user interface for computational packages in R [1] and repositories such as Bioconductor [2]. To address the interactivity limitation inherent in R libraries, our framework includes several native apps that provide exploration and brushing operations as well as an integrated genome browser. The apps can be chained together to create more powerful analysis workflows. To validate the usability of VisR for analysis of sequencing data, we present two case studies performed by our collaborators and report their workflow and insights.
4.1 Related work

We have discussed how sequencing technology can be used to measure the biochemical states of cells such as the expression levels of genes or binding sites of proteins in DNA. In Chapter 3 we saw how the ChIP-sequencing (ChIP-seq) data are used to analyze protein interactions with DNA. Another example is RNA sequencing (RNA-seq) which measures the presence and quantity of total RNA in a cell at a given moment in time and is widely used in gene expression analysis.

While computational methods to interpret these data continue to evolve, the rapidly changing computational tool set for data analysis often requires significant computational expertise to use. The Bioconductor project [2] is an open source software repository which hosts a wide range of statistical tools developed in the R programming environment [1]. Taking advantage of a rich set of statistical and graphical capabilities in R, numerous Bioconductor packages have been developed to address a variety of data analysis needs. The use of these packages, however, requires a basic understanding of the R programming/command language and an understanding of the documentation accompanying each package. As a result, R and the Bioconductor packages are primarily used by computer scientists and biologists who have a strong computational background, but remain inaccessible to most biologists who would significantly benefit from the ability to analyze such datasets. Hence, there is a clear need for a framework with an accessible user interface that allows biologists easy access to analytical tools for genomics data without requiring programming expertise.

Many useful tools have been developed for the visual analysis of biological data. What most of these tools have in common is that they have been designed to analyze and solve specific biological questions. Our goal was to push this envelop toward a more general-purpose visual analysis tool that could be applied to a broad range of analyses of sequencing datasets. This is not unlike such successful attempts as VTK [266], Prefuse [267], Polaris/Tableau [24], KNIME [34], Orange [31], Lyra [268] and CSIRO Workspace [269]. These systems attempt to bring data analysis through visual means to a large audience. Most of these tools also provide some form of integration with R and enable enriching their interactive data mining and visualization components with the statistical capabilities in R, however they are mostly accessible to users who have the technical skills for R development. In addition, they do not address the specific challenges associated with sequence analysis. Simple standards, such as integration of a genome browser or support for sequencing data are missing. Many of the tools for sequence analysis are meant to be used by bioinformaticians (as opposed to for biologists), and require programming skills. Those tools aimed at the biologists, on the other hand, offer limited analytical tools and are hard to extend or generalize.
4.1.1 R-based visualization systems

The lack of a graphical user interface (GUI) for the majority of the packages makes most of them inaccessible to biologists without programming expertise. Several frameworks have been developed to provide graphical user interfaces in R. Packages such as RGtk2 [270], fgui [271], R-Tcl/Tk [272], gWidget [273], JGR [274] and SciViews-R [275] allow programmers to create graphical user interfaces for command-line R packages. They have been used in general purpose packages such as Deducer [276], R Commander [277], GrapheR [278] and Rattle [279] as well as packages for biological data analysis, such as SeqGrapheR [278], limmaGUI [280], affyhmGUI [281] and OLINgui [282].

With the increased popularity of web-based analysis applications, several solutions such as Shiny [3], ggvis [283] and googlevis [284] have been developed to provide a web-based interface or an interactive implementation for R libraries. The graphical interfaces created by these libraries provide means to make the individual underlying R packages more accessible, however their scopes remain limited to the specific modules they are designed for and it is difficult, if not impractical, for biologists to link several modules to create more complex workflows. In addition, due to the significant coding effort required to create the graphical layout for each library and to pass the data to and from the GUI, most R libraries still remain without a graphical user interface.

4.1.2 Visualization systems for biological data analysis

In Chapter 2, we reviewed several visualization systems that mitigate the dependence of biologists on programmers and allow biologists to be more involved in computational analysis tasks. We reviewed genome browsers such as UCSC [118] and IGV [19] that allow users to navigate across the genome for detailed data inspection and exploration. While the genome browsers are useful for viewing specific genomic regions, they are not effective for global analysis and pattern discovery. Several systems such as CisGenome [113], seqMINER [198], Cistrome [253], EpiExplorer [201], Genomic HyperBrowser [285], and SeqMonk [114] have been developed to address the need for global pattern analysis. The strength of these tools lies in their ability to connect several analysis methods in a single application, but adding newly developed analysis pipelines is not easy and researchers may find themselves waiting for state-of-the-art algorithms to be implemented within these packages. A more recent related tool is Epiviz [206] that provides an interactive genome browser and data-analysis platform for functional genomics data. A scripting interface is also provided to invoke R functions and display the results within the tool, however this extension remains accessible only to users with relevant technical skills.
4.2 Requirement analysis

We held formative interviews with biologists from three centres (BC Genome Sciences Centre, UBC Life Sciences Institute and later UBC Biomedical Research Centre), to understand their analysis workflow and the limitations of their existing tools. In this section, we provide an overview of the tasks identified during the requirement analysis stage and then discuss our design process.

4.2.1 A typical analysis workflow

The initial task in a conventional sequencing data analysis is creating the data table for the regions of interest and sequencing data specific to the study. For each sequencing dataset, biologists compute a summary of the values of the sequencing data near each region of interest. The method for computation varies based on the type of the dataset and the study and can be as simple as adding up all values within the genomic interval, or more sophisticated methods involving machine learning (e.g. Hidden Markov Models or Baysian Networks) and non-linear normalization, but ultimately each dataset is generally summarized to one or multiple columns in a table.

Tools such as Galaxy [26] and SeqMonk [114] allow the creation of these data tables. These tools do a satisfactory job of helping biologists with the initial steps of data preparation such as quality control, sequence alignment, file format conversion and filtering. However they provide limited functionality for exploratory analyses and visualization. Thus far, such analyses can only be provided through additional programming interfaces / languages, such as R.

During the initial exploration phase, biologists frequently want to browse their datasets in a genomic context while studying the data table. The most popular approach for visualizing genome-scale data is to use genome browsers and displaying datasets as a histogram plots or heatmaps, on vertically stacked tracks. The exploration of data tables is often performed in common spreadsheet applications such as Microsoft Excel and involves sorting columns and looking and verifying the information at known regions of interest. Simultaneous use of the genome browser and table view is often a tedious task requiring switching back-and-forth between various applications while copy-pasting names or locations of genomic addresses from one application to the other.

Biologists employ a variety of computational methods from simple numerical calculations on the columns to more advanced generic or domain specific statistical or machine learning algorithms. Many biologists are comfortable doing the simple calculations supported by most spreadsheet software packages. However using more advanced techniques requires familiarity with programming or scripting environments, making them inaccessible to most biologists.

Results of the computations are then illustrated in plots such as histograms, bar charts, scatter plots and heatmaps. Based on those results biologists often repeat and iterate the
analyses with more refined subsets, for instance with rows for which a computed p-value is lower than a certain threshold.

### 4.2.2 Design

We followed an iterative user-centred design process. Based on the analysis of requirements we designed wire-frame prototypes, like the one shown in Fig. 4.1 created using Wireframe Sketcher [286], and presented them to our collaborators for evaluation. Once we were more confident about our designs, we proceeded to implementing interactive proof-of-concept prototypes using Java and APIs provided by the IGV [19] browser to read sequencing data formats. Our main rationale behind using a desktop platform (Java) as opposed to the a web platform (JavaScript) was being able to handle the inherently large sequencing data sets (gigabytes) while providing an interactive user experience, a similar rationale behind popular desktop genomic viewers such as IGV.

![Figure 4.1: Wire-frame prototype. A later stage wire-frame prototype created using the WireframeSketcher software.](image)

As we progressed through our design we realized that all three groups of our collaborators required a more general purpose system that was capable of solving several biological data analysis problems and flexible enough to adapt to new challenges. Our requirements eventually boiled down to:

- Not requiring programming skills to use
• Inherent support for sequencing data

• Integration of a base set of analysis methods such as dimensionality reduction (PCA, MDS), clustering (k-means, hierarchical), and RNA-seq analysis tools.

• Integration with a genome browser

• Ease of adopting new analysis methods (extensibility)

Figure 4.2: VisR evolution from initial mock-ups to interactive prototypes and released software

Fig. 4.2 shows the evolution of VisR from the initial prototypes to the final released software. In the following section we will discuss our design choices in more detail.

4.3 The VisR framework

In this section we will present the general framework and the design choices we made for VisR. We present the R-apps framework, which offers a semi auto-generated and unified graphical user interface for computational R packages and repositories such as Bioconductor [2]. We will then give an overview of the interface and its ability to chain apps together to create analysis workflows.

4.3.1 VisR apps

The functional units in VisR are called apps. A small number of the apps are interactive apps and are developed natively to allow interactive exploration and filtering. A larger number of the apps are developed in R and are called R-apps. To run an app, a user drag-and-drops the app icon to the workspace to create an active window for the app. The user
then specifies the input and parameters through the app’s graphical user interface. For interactive apps the app updates in real time as the user interacts with the active window or modifies the parameters. For R-apps, VisR will pass the user’s input data and parameters to the R environment, run the corresponding code and return the output results back to the workspace. The output can be images, data tables, columns appended to the input data table or files. The user can then use other apps to explore the results and link multiple apps to create more complex workflows.

4.3.2 R-apps

Every R-app consists of an R script file with the caller functions accompanied by a JSON file specifying the parameters to be passed to the R script. We had three main design goals when creating the R-apps. The first was to provide an accessible interface for biologists to use libraries in R, without requiring programming expertise. The second was to allow users to link the R-apps with the interactive components. The third was to minimize the effort required by R developers to create the R-apps user interface for new or existing R libraries.

At the core of an R-app is an R document which contains the required script to perform the desired functionality. It is up to the developer of the app to decide which parameters will be exposed to the user. These parameters will be assigned unique parameter names (exposed to the user of the app and therefore can be different from internal names). The parameter names and their types are then placed in a file in JavaScript Object Notation (JSON) format with the same name prefix and with extension .json.

Table 4.1: Supported types for input parameters. Variable type definition keywords, corresponding R data types and the generated GUI component.

<table>
<thead>
<tr>
<th>variable type</th>
<th>R data type</th>
<th>GUI component</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer</td>
<td>JSpinner</td>
</tr>
<tr>
<td>double</td>
<td>numeric</td>
<td>JSpinner</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>JCheckBox</td>
</tr>
<tr>
<td>string</td>
<td>character</td>
<td>JTextField</td>
</tr>
<tr>
<td>string with items</td>
<td>character</td>
<td>JComboBox</td>
</tr>
<tr>
<td>filename</td>
<td>character</td>
<td>JFileChooser</td>
</tr>
<tr>
<td>color</td>
<td>character</td>
<td>JColorChooser</td>
</tr>
<tr>
<td>range-int</td>
<td>vector</td>
<td>MyRangeSlider</td>
</tr>
<tr>
<td>range-double</td>
<td>vector</td>
<td>MyRangeSlider</td>
</tr>
<tr>
<td>column</td>
<td>matrix</td>
<td>JComboBox</td>
</tr>
<tr>
<td>column-numerical</td>
<td>matrix</td>
<td>JComboBox</td>
</tr>
<tr>
<td>multi-column</td>
<td>data.frame</td>
<td>JList</td>
</tr>
<tr>
<td>multi-column-numerical</td>
<td>data.frame</td>
<td>JList</td>
</tr>
<tr>
<td>output-column</td>
<td>vector</td>
<td>JTextField</td>
</tr>
<tr>
<td>output-table</td>
<td>data.frame / matrix</td>
<td>JTextField</td>
</tr>
</tbody>
</table>
Table 4.1 shows the current supported variable types, the corresponding R type and the generated GUI component. An optional icon can also be specified by providing a .png file with the same name prefix. Once VisR starts, it searches through a specific directory for all *.R files with an accompanying .json file and populates the Apps pane in the main user interface. A default gray box is used as the apps icon if an image with the app’s name is not found. When the user drags an app into the workspace, the app’s .json file is parsed and the graphical user interface is automatically created using Java’s Swing library. In addition to providing a unified user interaction model, our intention was to minimize the effort required by developers to create apps. Unlike the previously mentioned related work on creating user interfaces for R, which required users to write the code for the actual graphical interface, we have kept the requirements to the minimum of specifying the input parameter names and types.

Once the user specifies the parameters and hits the Run button, an R session is created using the Rserve [287] library. Rserve is a TCP/IP server which allows client programs to use facilities of R from various languages including Java without the need to initialize R or link against R library. The input data table and user specified parameters are passed to the R session and the R code is executed line by line. The textual output of the R is directed to a console pane and the final graphical output is displayed in the pane assigned to the specific app. A progress animation is displayed inside the app’s pane while the code is running and the user may terminate running the app by pressing the cancel button.

Apps may also have output variables. Currently we support column, table or file output. If the user specifies a name for the output (i.e. the name for the column, table or file), the output of the app is read back from the R session. A user may specify a new name to create a new column, table or file or use an existing name to overwrite one. These outputs can also be used as inputs in other apps, allowing the users to link several apps.

In addition to the auto-generated GUI, more experienced users may also browse and modify the R code by selecting the “Code” tab above the parameters pane. This will show a syntax highlighted text editor with the R code that can be edited and executed within the tool. While this is not meant to be a full featured R development environment such as RStudio [288] it is useful for more technical users as a quick way of browsing the R code and making small modifications to the apps without requiring to exit the tool.

As mentioned, our goal is to minimize the effort of R developers to create R-apps. Thus the information required to create the GUI is kept to the minimum of specifying the variable’s name and type (in fact specifying the type is also optional when the input is a string). However the app developer has the option to enrich the interface by specifying the following additional information:

- **categories**: grouping variables together. They can be collapsed or expanded by default.
• **label**: specifying the label shown in the GUI. If not specified, a label will be generated from the variable name by replacing the underscore "_" characters with space " " and removing the "input" prefix, if any.

• **info**: specifying details about the variable to be shown as a tool tip text.

• **default**: specifying a default value for the variable displayed in the initialized GUI.

• **min / max**: specifying the valid input range for the integer and numeric variables.

• **items**: showing a list of string items to choose from.

• **ui**: customizing the user interface. Currently, this is only implemented for file variables where specifying "load" or "save" will create a load or save dialog box. Additional options are planned to be added to the system to add more customization to other variable types, such as choosing between a spinner or slider for numerical columns or between combo box and radio groups for items.

### 4.3.3 A simple R-app

To show the simplicity of creating R-apps we walk through a simple 2D plot that uses R’s default plotting functionality. Fig. 4.3(a) shows the R code for a simple 2D plot. It takes two required parameters, `input_x` and `input_y`, the column names used for x and y, and three optional parameters, `input_color` for the point colors, `input_log` for selecting logarithmic scale and `input_title` for the plot title.

Fig. 4.3(b) shows the input parameters specified in JSON format. The type specified for `input_x` and `input_y` is `column_numerical` which indicates the GUI should list only the numerical columns of the input table, while the type of `input_color` is specified as `column` so any column is a valid selection. For `input_log` a list of four strings ("", "x", "y", "xy") is specified with the first one being the default. The type for `input_title` is not specified so it will be considered a string input by default. The graphical user interface generated from the parameter specification is shown in Fig. 4.3(c) and the graphical output of running the app with example input parameters is shown Fig. 4.3(d).

While developing apps for the VisR framework, we realized that using the JSON syntax to manually specify the parameters had two issues: first, the JSON syntax was not concise enough and it was easy to make mistakes specially for larger apps; and second, it was difficult to debug and test the R-app, outside the VisR framework. So we created an API that allows generating an app’s JSON file as well as initializing the variables within the app’s code so the app could also be executed and debugged outside VisR, for instance in a development environment such as RStudio. An example R code to create an R-app is shown in Fig. 4.4(a), and the corresponding user interface generated for the app in VisR is shown in Fig. 4.4(b).
Thus far, we have implemented close to 50 analysis and visualization apps as well as widely used packages from the Bioconductor project. Among those, are:

- several apps for general plotting including scatter plot, bar plot, box plot, heatmap, violin plot, Circos [14], UpSet [289], etc.
- apps for clustering including $k$-means, $k$-modes, DBSCAN [290], hierarchical clustering, and spectral clustering.
- apps for dimensionality reduction including PCA, MDS, and tSNE.
- several apps for differential expression analysis including DESeq [291] and EdgeR [160]
- methylKit [193] and DMRcaller [292] for analysis and visualization of DNA methylation profiles
- Seurat [214] and Monocle [217] for single-cell analysis

Prototyping each app took less than an hour for simple apps such as the PieChart and BarPlot apps and a few hours for apps that required going through the package’s documentation and samples. These approximate times are just for the initial creation of the apps with basic functionality and we put extra efforts over time iterating on each app with the users to improve the usability or to add new functionality.
Figure 4.4: Using the R API to specify parameters for an R-app: (a) The R code specifying the app’s name, the parameters, and the code to perform the desired functionality. (b) The user interface generated for the app in VisR.

It is worth mentioning that while we were aware of the inferiority of some of the visualization techniques (e.g. pie charts and Venn diagrams) we included them as they were requested by our collaborators and used in their workflows.

4.3.4 Native apps

Native apps are created in Java and are pre-compiled with the framework to allow interactive graphics. The basic mental model of our views is a table that ties all views together. However, during our requirement analyses, our users frequently asked for an interactive interface for some of the apps to allow interactive navigation as well as brushing operators to select subsets. Since R doesn’t provide such interactivity, we realized the tool would not be completely useful without interactivity at least for basic plot types. The ones with popular request were histogram, scatter plot, Venn diagram and a genome browser.

**Table View:** displays a data table in a layout common in spreadsheet software (Fig. 4.5(a)). Clicking a column header shows a popup menu allowing users to perform several tasks such as sorting the table by that column, removing, or editing columns (e.g. changing the equation for calculated columns).

**Histogram:** provides a standard interactive frequency plot (Fig. 4.5(b)). Any numerical table column can be used for the x-axis. The y-axis can have optional transformations such as log, cumulative distribution function and percentage. Users can perform standard panning and zooming interaction or directly specify exact values. The histogram plot offers a range filter that can be used to create a subset of the rows with their value falling within the range. Users can choose to have more than one segment for each range filter and specify
whether the ranges should be equally spaced or have equal number of items. Once a filter is created it persists for that table and it is updated whenever the data values change.

**Scatter Plot**: shows an interactive 2d scatter plot (Fig. 4.5(c)). Users can specify multiple columns to the horizontal or vertical axis to effectively create a scatter plot matrix. Users can select a group of points and create a subset using the rectangle, polygon and quad filter provided. Similar to the histogram range filter, the scatter plot filters will persist and update as the data is changed.

**Parallel Coordinates**: shows an interactive parallel coordinates plot. Users can specify the axis to be visible in the plot and specify visual properties such as colors and transformations to be applied to the data items.

**Venn diagram**: shows approximate area preserving Euler diagrams or symmetric Venn diagrams (up to 5 sets) for the subsets assigned to the plot (Fig. 4.5(d)). Users can toggle between the two modes. The transition from one mode to the next is animated. The diagram is updated when any of the subsets change.

**Genome browser**: we integrated IGV as a widely used genome browser (Fig. 4.5(e)). Users can load the tracks that are displayed in the genome browser into their data tables. For tables that have columns with genomic location, clicking on the rows in the Table app or on the points in the Scatter plot app navigates the Genome Browser to the corresponding genomic location.
4.3.5 The VisR graphical user interface

The VisR graphical user interface is shown in Fig. 4.6. It is split into several panes exposing the different functionalities provided in the framework. At the left-hand is the Data Pane (Fig. 4.6(a)) which depicts current data tables loaded in the system. The right-hand panel contains the Apps pane on the top (Fig. 4.6(b)) and the Parameters pane on the bottom (Fig. 4.6(c)). The Apps pane contains the icons for the modules available to the user; we will refer to them as “apps” throughout the rest of this chapter. The Parameters pane shows the input parameters for the currently selected app. At the center is the workspace area (Fig. 4.6(d)), where the panes for the currently running apps are laid out. Each pane is customized based on the utilities of each app, but for most apps it displays a graphical output. The textual output of apps is displayed in the Console pane at the bottom of the workspace (Fig. 4.6(e)).

Fig. 4.6 shows an example layout after some analysis steps. Users may change the layout of the panes to customize it based on their display size or workflow requirements. For example, we observed some users preferring to overlap the parameters and apps pane into a tabbed pane to utilize the entire horizontal space when specifying parameters. In the following sections we will describe the interface in more detail.

**Data pane**

As previously mentioned, VisR’s default internal data model is tabular: a collection of records with named attributes of a given data type. Users can create tables either from
the feature files containing genomic regions or load text files in comma separated format. During analysis, subsets may be created through filtering, preserving the inherit hierarchy of these sets.

Table columns are either data columns created from sequencing data, calculated columns, or output columns of apps. VisR provides an interface with a variety of options to process and normalize sequencing data in aligned read formats (e.g. BAM [293]) and continuous-valued formats (e.g. WIG [244]). This was one of the first features in the working prototype and was much appreciated by our collaborators as it enabled them to use their own as well as available public datasets for their analysis. Inspired by the calculation option in most spreadsheet software, we added a calculator interface using a similar syntax to Microsoft Excel that our collaborators were well familiar with. As discussed previously, R-apps may also be used to create new columns or overwrite columns of an existing table (e.g. a computed cluster id or p-value).

**Apps pane**

Apps are the analysis modules of VisR. The Apps pane hosts an iconic view of the available apps. Individual panes for any App are created by dragging the app’s icon and dropping it at the desired location in the workspace. A highlight box shows the placement of the new pane as the user drags and moves the app over the workspace area. Once an app pane is added to the workspace, the user assigns the input table to the app by dragging the desired data node from the Data pane into the app’s pane. The user can click the cloud icon to download the latest version of the R-apps from the default repository\(^1\) or any other repository specified by the user.

**Parameters pane**

Whenever the user clicks on the output pane of an app, the parameters pane is updated to show the parameters for the app. As previously explained, the user interface for the parameters of R-apps is automatically generated from a JSON file describing the input and output variables. We initially had the parameters within a popup dialog, but that made it hard for the user to incrementally tweak the parameters and see the results, especially for the interactive plots. We then placed the parameters side by side with each app, but realized this was an inefficient use of screen space, especially since users were usually modifying the parameters for a single app at a time.

\(^1\)https://github.com/visrsoftware/visr-apps
Workspace pane

The panes in the workspace are laid out in a tabbed/tiled document interface similar to the layout system in rich client platforms (RCP). Our initial prototypes used a multi document interface (MDI) with fixed position for the default panes. Through user evaluations we noticed that the workflow frequently became cluttered, making it hard to organize and find the open apps. Changing to an RCP interface took time for our test users to get comfortable with, but then they expressed satisfaction with its flexible layout and how it allowed them to keep their workflow organized and clean.

Console pane

The console pane was required to show the textual output, progress and error messages of the apps. Since multiple R-apps may be running together, we only show the textual output of the currently selected app.

4.3.6 Limitations

VisR provides a simple way to link multiple apps, however it is currently limited to libraries that use R’s standard data types as their parameters and output. More complex data types such as complex tables and custom classes cannot be integrated directly through the current interface options. However it is possible to use files as a connecting medium the approach which we have used in the single-cell apps.

We were not able to fully automate the creation of the apps from the Bioconductor packages due to the large variety in the interfaces and input parameters for the these packages. There is no standardized meta data provided with these packages, however text mining methods might be a possibility to explore to extract those meta data from the user manuals. Still, in comparison to the previous GUIs for R, we believe we have significantly reduced the extra work by only requiring the parameter types and the function calls for each library to generate the GUI and link between different libraries in an analysis workflow.

In terms of scalability, our users have typically dealt with datasets of tens of thousands to a few million data points. Some users have worked with about 30 million data points (one data point per 100 base pair for a 3 billion base pairs genome). The framework has been robust to handle these cases, however the interactive apps become less responsive for data sets with more than few million data points. The responsiveness of R-apps depends significantly based on the computational complexity of the implementation of the corresponding R packages. A simple box plot of 30 million data points takes 10-20 seconds on a typical personal computer, while a hierarchical clustering can take a few hours to finish.
4.4 Evaluation

In our work, we differentiate between biologists and bioinformaticians. Biologists have the knowledge to analyze the sequencing data. However, they often do not have strong programming skills to use computational tools that only have a scripting interface. Bioinformaticians on the other hand have a strong algorithmic training and enough familiarity with the problem domain to develop computational tools for biologists. However, they often do not have the biological understanding required to analyze the data. The former are the target end users of this tool and the later are most suited to develop new apps.

To validate the usability of the tool to achieve the design goals, we conducted several case studies with collaborators who were interested in analyzing such data sets in their laboratories, two of which are presented in this section. Additional documentation as well as several workshops on single-cell analysis are available in the user guide section of the website.

4.4.1 Case study 1: gene expression in stages of mouse development

For this case study our collaborators were studying RNA-Seq data from cells from two stages of mouse development: embryonic stem cells (mESCs) and primordial germ cells (PGCs). For each development stage cells lacking expression of specific genes (knock-out/KO) as well as “control” cells with normal expression (wild-type/WT) were analyzed. Our collaborators specifically selected KO and WT datasets for four genes, “SETDB1”, “KAP1”, “G9a”, and “HP1” to investigate the overall transcriptional correlation among cells lacking expression of these genes and the corresponding WT controls. In addition, they were also interested in identifying the genes that were up or down-regulated in knock-out cells compared to their corresponding wild-type controls. It was previously shown that these four genes play a role in the deposition of certain repressor epigenomic modifications [294, 295, 296]. Therefore, lack of expression of these genes is likely to alter the transcription of many genes. Because mouse strains are genetically diverse and each biology lab typically uses just one or a small number of the available strains to conduct their experiments, a strain-specific heterogeneity is observed among wild-type mice which complicates the comparison of KO cells derived from different mouse lines. So our collaborators were interested in characterizing this heterogeneity among those four wild-type mESC lines derived from different mouse strains.

Analysis 1: Identification of transcriptional correlation

Our collaborators started by using the data import functionality to generate a data table containing the 10 RNA-seq datasets, using the RPKM (reads per kilobase per million mapped reads) statistic [297] for normalization. Subsequently, they used the “correlation” R-app, and chose the Pearson coefficient option to calculate the correlation of expression profiles among these 10 columns. The result, a $10 \times 10$ correlation matrix, was added to the
Data pane as a new data table. To visualize the correlation, they used this new table as the input of the heatmap and the MDS R-apps. In Fig. 4.7(a), the heatmap output pane displays a heatmap visualization with hierarchical clustering on the correlation values illustrating the biological difference of these samples based on expression profiles of all genes. The MDS output pane added two new columns to the correlation table as the result of multi-dimensional scaling. These two columns were then used as the input of the R scatter plot app (Fig. 4.7(b)). Both visualizations showed that the overall gene expression profile for each KO sample is more similar to its corresponding WT than any other KO sample. In addition, PGCs show significant dissimilarity of gene expression to mESCs.

**Analysis 2: investigation of differential expression**

While the overall gene expression of KO and WT cells were fairly similar, our collaborators were interested in exploring the small fraction of genes which were up or down-regulated in KO compared to WT cells. To identify such genes, they used the R-apps for two Bioconductor packages, DESeq [291] and edgeR [160], which are the state of the art methods in the genomics field for conducting differential expression analysis. Both of these methods show their best performance when biological or technical replicates exist for RNA-seq samples. Since our collaborators did not have biological replicates for any of their RNA-seq samples, they created four random sample sets for each dataset containing 30% of the data and used it as a technical replicate. They were curious to compare the results of these two methods with and without technical replicates. To do a fair comparison, our collaborators selected the Benjamini and Hochberg algorithm [298], available in both DESeq and edgeR R-apps, to generate the false discovery rate (FDR) for each gene and applied an FDR threshold of 0.01 to identify genes showing significant changes. The output for each app was a new column indicating the predicted cluster id for each gene: “+1” for up regulated, “-1” for down regulated and “0” for no difference. The result of the two methods, DESeq and edgeR, on the data without and with replicates, are shown in Fig. 4.7 (c-f). The bar plots show the size of each predicted group and the scatter plots show the genes colored by their predicted group, on a log scale of the normalized gene expression value in the wild type (HM1) vs. knock out (HP1aKO) cells. Our collaborators then used the interactive scatter plot app to create 8 subsets of up and down regulated sets (“+” or “-”) for each of the two methods (“D” or “E”), with (“R”) and without replicates (Fig. 4.7(g)). These subsets were then compared in the Venn diagrams shown in Fig. 4.7(h-k). As shown, the edgeR result is relatively more robust in terms of the genes that are identified as up or down-regulated in HP1aKO compared to WT, whereas the DESeq result changes significantly when technical replicates were used. However, the number of genes reported by edgeR as differentially expressed genes in HP1aKO compared to WT is at least three times more than their counterparts in the DESeq analysis. This indicates that edgeR is more sensitive to the outliers, as reported previously [299]. Because the high specificity was more important than high sensitivity for
Figure 4.7: Workspace of gene expression analysis in case study 1. (a) Heatmap with dendogram plot, and (b) MDS plot, showing the transcriptional correlation. (c-f) scatter and bar plots showing the genes up-regulated (blue color and labeled “1”), down-regulated (red color and labeled “-1”) or non-differentially expressed (green color and labeled “0”) based on DESeq and edgeR R-apps with replicates (“DR” and “ER”, respectively) or without replicates (“D” and “E”, respectively). The results for pair-wise comparison of DR, ER, D, and E runs have been shown as Venn diagrams for up-regulated and down-regulated genes. (g) Native scatter plot app showing the intersection between the result of DR (DESeq with replicates) and D (DESeq without replicates) runs for up (“1”) vs. down (“-1”) regulated genes. The rectangle filter in native scatter plot is used to create the four subsets for Venn diagrams (h-k).

Our collaborators, they chose to use DESeq for their differential expression analyses because of the low false positive rate of DESeq shown by these empirical results.
4.4.2 Case study 2: allele-specific gene expression

Each diploid cell consists of two copies of the genome, one from each parent (haplotype genome). In inbred mice, these paternal and maternal copies of the genome are identical. Hybrid mice, on the other hand, can be derived from crosses between distantly related laboratory inbred mouse strains, which differ at numerous genomic loci. Our second group of collaborators used a recently published dataset for trophoblast cells from hybrid crosses between CAST/EiJ (Cast) and C57BL/6J (B6) mice [300] and generated allele-specific (AS) profiles using the ALEA pipeline tool [301]. They were interested in a quantitative analysis of their allele-specific (AS) profiles to identify genes in RNA-seq data and/or genomic regions in ChIP-seq data showing allelic skew in one haplotype vs. the other.

Analysis 1: characterization of genes with allelic imbalanced expression

Our collaborators started by using the DESeq R-app on the AS RNA-seq data, the output of which was displayed using the R scatter plot app shown in Fig. 4.8(a). DESeq identified 438 genes with allelic expression skew toward CAST and 55 toward B6 haplotypes. A subset of these genes were so called “imprinted genes” [302], which are known to show mono-allelic expression.

Our collaborators were curious why the number of genes that are highly expressed in CAST but not B6 was significantly higher than the number of genes that are expressed in B6 not CAST. So they opened the interactive histogram app and used the range filter to create the subsets for genes with mono-allelic expression for CAST and B6 Fig. 4.8(b).

Plotting the two subsets in separate pie charts, revealed that the majority of genes (> 350) with allelic expression skew toward CAST are located on the X chromosome (Fig. 4.8(c)), but there is no visible pattern for B6 (Fig. 4.8(d)). This was expected as the X chromosome is known to host a large number of maternally expressed genes (expressed in CAST haplotype here) through X chromosome inactivation [300].

Analysis 2: exploration of allele-specific relation between H3K36me3 and gene expression

The epigenomic modification H3K36me3 was previously shown to be enriched in the gene body of active genes [303]. To support the above AS analysis workflow for RNA-seq data, our collaborators decided to study the potential relation between AS profiles in RNA-seq and H3K36me3 data for the genes showing allelic imbalanced expression. The CAST allelic contribution in both RNA-seq and H3K36me3 data was calculated using the tool by dividing the allelic read counts assigned to CAST to the total number of allelic reads per gene. As shown in Fig. 4.8(e), all the 438 genes characterized in the RNA-seq analysis by DESeq as candidates with high CAST allelic contribution, show the same pattern for H3K36me3. In contrast, the genes having low CAST contribution (high B6 contribution) in RNA-seq
data do not necessarily show low CAST contribution in H3K36me3 data. An IGV browser view of AS profiles for H3K36me3 and RNA-seq data (Fig. 4.8(f)), shows two of the known imprinted genes “Slc38a4” and “Mirg”, are expressed in a mono-allelic manner in B6 and CAST respectively and concurrently enriched with H3K36me3.

Taken together, this case study showed the potential of VisR as a visual analysis toolbox to enhance other bioinformatics tools such as ALEA by providing a visual interface to required statistical packages from R.

4.5 Conclusions and future work

In this chapter we presented VisR, a framework for analyzing sequencing data and creating interfaces for R libraries. By reducing the required technical expertise, VisR facilitates data analysis for a broader set of biologists and bioinformaticians. We created a small but diverse set of R-apps to demonstrate VisR’s utility and flexibility. We also provide several native
apps to support interactive exploration of the data together with the output of the analysis methods. VisR is now being used actively by our collaborators, who report that they are able to perform their analysis pipeline more quickly and efficiently than with existing tools.

There are a number of directions for future work. We have begun to introduce VisR to other labs in addition to our initial collaborations. We are currently observing the use of VisR by these new users to evaluate and improve the usability and effectiveness for more diverse and complex analysis problems. Labs which were especially keen on working with VisR are employing students with bioinformatics background in order to adapt their existing R-based custom analysis modules into R-apps making them more accessible to more lab members. In addition to making some of these modules available in a future release of VisR, this has helped us improve the process of developing new apps.

Several features are still required to support the usability of our tool. For workflows to be truly useful, there must be clear and repeatable records of what has been done, like the history system offered by tools such as the Galaxy platform. Other features include saving and loading the workspace together with the parameters, and an undo possibility. Further, reusing common parameters between apps, such as graphical variables shared among different plots, would improve the usability of the tool.
Chapter 5

A general purpose computer model exploration system

In Chapter 4 we discussed VisR, our extensible and interactive framework for visual analysis of sequencing data. The rapid development API of VisR allowed us to create over 50 R-apps in a short period of time and the usage statistics show a growing number of users since the release. However, even though the graphical user interface of the R-apps allowed non-programmers to utilize the computational power of the R environment, a new challenge surfaced. Most apps have several (typically between 3 to 5, but sometimes over 10) numerical or categorical control parameters. Although the parameters for each app were initialized with the default values suggested in the corresponding R package and some documentation was provided in the form of tool tips, finding the “right” parameters for each use case involved a tedious manual trial-and-error approach. Encouraged by the previous successful attempts with visual parameter exploration tools we implemented an extension of VisR to support general visual parameter space analysis for R-apps.

In this chapter, we will present our system, called ModEx, that can be used for exploring parameters of a variety of computer models. We will evaluate the effectiveness of our system by demonstrating it in different application domains including differential gene expression analysis.

5.1 Introduction

Many complex computer models require users to provide several parameters to tune the output based on the specific computational needs. Such models appear in a variety of modeling approaches, such as data science (regression, classification, and clustering) but also computational science (numerical modeling). Today, there is almost no field of science untouched by the application of such modeling techniques. In many applications, fully automated optimization is not possible due to the trade-offs between the often contradicting objectives. A classical example is the trade-off between precision and recall for classification problems,
where a higher value of one leads to a lower value for the other. This requires a domain expert to inspect different models (or results of the modeling process) to make an informed decision. Manual trial and error of running models with different parameter settings is tedious and usually requires many iterations involving guess-work and luck. In addition, the required effort for trying different parameters increases exponentially with the number of the parameters of the model. This not only makes the exploration of the parameter space extremely inefficient, it also prevents a global understanding of the model capabilities.

To address this, several systems and workflows have been developed to automate the process. These solutions run the model with different combinations of parameter values sampled from parameter space. This is done in a preprocessing step and the results are collected. Then a visual exploration interface is used to systematically explore the parameter space and further analyze the results. Many of the systems that support these types of “visual parameter space analysis” (vPSA [304]) techniques are built for various modeling approaches and applications domains such as simulation [305, 306, 307, 308, 309], image analysis [310, 311, 312, 313], data mining [314, 315], industrial decision making [316, 317, 318] and bioinformatics [319]. These solutions are custom-made and developed for specific application domains. The cost of building these tools is usually high and often involves several months of development work. As a result, parameter exploration solutions lack a wide-spread adoption for many computer models that can benefit from the approach.

In this chapter we discuss our general purpose system that facilitates visual parameter analysis for a wide variety of computer models. Our system offers key components of a visual parameter analysis framework including parameter sampling, deriving output summaries and an exploration interface, which can work with a wide range of tools and libraries minimizing the required setup time. It also provides a flexible API for rapid development of parameter space exploration solutions for custom application domains. We also demonstrate this flexibility and effectiveness in three application domains: data mining, machine learning, and gene expression analysis.

As part of our vPSA system which will be introduced in Section 5.3, we make three primary contributions. First, we present a novel method of creating user interfaces suitable for specifying the relevant parameter space and parameter sampling. These interfaces are created automatically and seamlessly for computer models wrapped into modules called apps using a parameter description API. Second, we present a generic approach to handle derivations of outputs and show how it can be extended for custom application domains. Our third primary contribution is the design and implementation of a visual exploration interface to allow analyzing the results of parameter sampling, either to find “right” parameters for a specific problem (for model users), or to get a deeper understanding of a computer model’s inner workings (for model developers). The user interface is designed to be flexible and to allow creating custom workflows by linking either existing computational modules or creating new ones.
Our secondary contribution is an evaluation of our system in different application domains. We present three case studies: In Section 5.4.1 we consider the problem of clustering and demonstrate how ModEx can be used to quickly recreate the main functionality of similar state-of-the art tools for comparing different cluster algorithms. In Section 5.4.2 we study the problem of hyperparameter tuning for neural networks and use a popular educational neural network application as our model for ModEx. In our third case study in Section 5.4.3, we use ModEx to analyze two widely used bioinformatics methods for differential gene expression analysis.

5.2 Related work

Sedlmair et al. [304] introduced the conceptual framework for visual parameter space analysis (vPSA). They identified a dataflow model with the essential components of the model, a surrogate model as well as the derive step. Their work was based on 21 papers from the visualization community. This was the main inspiration of our work and we followed their guidelines. We are supporting the model component as well as the derive component. However, currently we do not support the surrogate model—only 5 of the 21 papers analyzed by Sedlmair et al. used such a component. Still, such a component is of importance but is left as future work for now.

5.2.1 Applications in machine learning

The terms Machine Learning as well as Data Mining are often considered to be rather broad terms encompassing techniques in regression analysis, classification analysis, clustering, outlier detection, and dimensionality reduction. In this chapter, however, we consider the focus and contributions of machine learning to be mainly in the area of classification analysis and the contribution and focus of data mining to be mainly in the area of clustering analysis.

As far as classification is concerned, TreePOD by Mühlbacher et al. [320] supports a user in understanding the trade-offs of various decision trees. Through a parameter sampling approach, a large number of decision trees are built. They are then evaluated based on accuracy in addition to aspects such as ease of understanding by decision makers. The user is then able to inspect relevant trees that are on the Pareto frontier of such trade-offs.

One of the most promising approaches for machine learning today are deep neural networks [321]. There has been a lot of attention paid to the analysis of particular deep neural networks. One of the hardest challenges considered today is the understanding of a proper network architecture, or the determination of proper hyperparameters. Recent surveys by Liu et al. [322] and Hohman et al. [323] focus on visual analytics approaches, however, to our best knowledge there is no generic vPSA framework solution addressing this problem.
5.2.2 Applications in data mining

In data mining and particularly clustering, Kwon et al. [315] presents a comprehensive discussion of clustering methods and visualization systems for cluster analysis and discusses several categories of systems. One group are “visual analytics systems that employ clustering as a part of high dimensional data analysis” which include Hierarchical Clustering Explorer (HCE) [324], VISTA [325], and DICON [326]. A second group are those which “allow users to provide feedback on clustering results so that the next run applies their inputs” such as desJardins et al. [327], iVisClustering [328], Cluster Sculptor [329], Boudjeloud-Assala et al. [330], and Clusterix [331]. A third group are those which “allow users to generate and compare multiple clustering results with respect to their quality”. Example research falling in this group are Turkay et al. [332] and XCluSim [333].

5.2.3 Applications in bioinformatics

As mentioned earlier, our initial motivation for developing ModEx came after our collaborations with biologists to develop methods for understanding biological data. Most of the bioinformatics research has focused on statistical and computational methods for processing and analysis of the biological data. Some of the most used ones are limma [334], edgeR [160], and DESeq [291] which will be further discussed in our case study in Section 5.4.3.

There have also been several works done to provide guidance in selecting the suitable algorithms and parameters for those computational methods based on the dataset and the goals of the analysis. Among them, a group that are most related to this work are approaches that combine cluster analysis with interactive visualization techniques to facilitate analysis and understanding of large data. Examples of such methods which provide visual interfaces for tasks such as comparison of several clustering results are Genesis [335], HCE [324], Mayday [170], XCluSim [333], MLCut [336], VisExpress [169] and Kern et al.[337].

5.2.4 Definitions

The terminology used throughout this chapter, closely follows the guidelines in the vPSA conceptual framework [304]. For clarity, we will briefly explain some of the most frequent used terms.

- **Computer model** or simply the model, is the algorithm or set of algorithms to which the vPSA system is being employed. In this chapter with computer models we mean statistical models as it is common in data science or simulation models as it is common in computational science. The terminology used in the vPSA conceptual framework is *computational input-output models*.

- **Run** is one execution of the model with specific parameter values.
- Parameter combination is the randomly generated set of values used as control parameters in a single execution of the model. Different runs of the model will often use different parameter combinations.

- Derived output are objective measures that summarize the essential characteristics (e.g. quality) of the complex model output.

5.3 A general purpose system for visual parameter space analysis

This section introduces our general purpose system for visual parameter exploration of computer models. ModEx is built as a component of the VisR framework, utilizing its API for creating R-apps and interactive apps. We were initially focused on addressing the challenges in the bioinformatics application domain. However, through the development process we realized that most of the analysis tasks are generic and applicable in other application domains. As such, ModEx has iteratively evolved to be usable in a wider variety of application domains. We will demonstrate this generality in Section 5.4.

5.3.1 Design goals

From the very early phases, an essential design goal was to develop ModEx such that it follows the same design principles of VisR to avoid alienating the existing user base. We also wanted to utilize the methodology of vPSA for existing R-apps without requiring significant development. In addition, we wanted the possibility of utilizing ModEx for models not implemented in R. We will elaborate our design decisions based on these goals in the remainder of this section.

5.3.2 Parameter sampling

Our first step was to extend the existing automated GUI system so that it could also be used to specify a range (or space) of parameter combinations.

Fig. 5.1A shows the generated standard (single parameter) view which is the default when a user adds an app to the workspace. In a typical scenario, a user specifies the parameters through the standard view and then clicks on the “Run” button (Fig. 5.1a). VisR then passes the user specified parameters to the R environment, executes the app’s code, and returns the results back to the workspace for further exploration. In ModEx, a user can now click on the toolbar button shown in Fig. 5.1b to toggle between the standard view and the ModEx view mode shown on Fig. 5.1B. In the ModEx view mode the user interface controls are generated such that they allow specifying ranges of valid values for the parameters as follows:
Figure 5.1: The GUI generated from the parameter specification in Fig. 4.4: (A) Standard view, and (B) ModEx view. (a) run the R-app. (b) switch the GUI to ModEx view. (c) number of times to run the app. (d) start running the R-app for the specified number of runs. (e) directory to store the output of runs.

- **integer/double**: integer/double lower and upper bounds for the parameter sampling range. The acceptable minimum and maximum can be specified in the app’s declaration script (see Fig. 4.4).

- **boolean**: two check boxes with options yes and no

- **string** with defined items: a check box for each item.

- other parameter types: such as generic string, filenames, or output parameters will not require custom made UI controls.

UI controls are initially set to the default value specified in the app’s declaration script. At that state, the value for the corresponding parameter will stay constant during the parameter sampling. This allows users to choose which parameter values are kept unchanged and which ones are sampled randomly from the specified range. Currently only random uniform sampling is implemented and exploration of other sampling methods is discussed.
as a possible future work in Section 5.5.1. The total number of parameter combinations can be modified by changing the value of “Number of Runs” shown in Fig. 5.1c. We initially had the option to specify either the number of runs, or an end time when sampling should be stopped. However using end time turned out to be inconvenient as different computer models had significantly varying execution times which could result either in too many or too few number of runs. We instead decided to allow users to stop any time without corrupting the final output. They can also resume the runs at a later time if they later realize more samples are required. This allows partial and off-line (as opposed to real-time) support for the *simulation steering* strategy explained in the conceptual framework where the user can make adjustments while the computer model runs, for example, to change some input parameter settings.

![Diagram](image.png)

**Figure 5.2:** The ModEx data flow: (A) Parameter Sampling step; (B) Derivation step. Parameter sampling includes: (a) the computer model / app; (b) input data to the app; (c) parameters of the app; (d) runsInfo table containing information about each run, e.g. parameter values; (e) data outputs of the app; (f) image outputs of the app. Derivation includes: (g) derivation methods; (h) derived output (in memory); (i) derived output added to the runsInfo table.

The parameter sampling will start when a user clicks on the “Start Sampling” button (Fig. 5.1d) and the results are stored in the user specified directory (Fig. 5.1e). The sampling process is demonstrated in Fig. 5.2A and includes the following:

- *input.txt*: the data table used as the app’s input, shown in Fig. 5.2b.
• *paramInfo.txt*: a data table containing the name and type of the app’s parameters, shown in Fig. 5.2c.

• *runsInfo.txt*: a data table with each row corresponding to a single run and columns containing the values for the parameter combination, as shown in Fig. 5.2d. In addition to the parameter combination values, each run is also assigned with a unique integer ID used to reference the output for that run.

• *<runs>*: subdirectory containing all output from the runs, each suffixed with the run ID. For instance, if the R-app outputs two data tables, TableA and TableB, the *<runs>* directory will contain TableA_0.txt, TableB_0.txt, TableA_1.txt, TableB_1.txt, and so on, such as the example shown in Fig. 5.2e.

• *<images>*: subdirectory containing image outputs from the runs, named as *[ID]*.[ext], e.g., 0.png, 1.png, and so on, as shown in Fig. 5.2f. This can be extended to include other complex objects.

This directory structure is the same for all R-apps sampled in ModEx. To use ModEx with none-R models, there are two alternatives: If the model has a command-line interface, we can create an R-app that includes the parameter specification, but utilize the R’s “system” function [338] to invoke the none-R tool with the user parameters. The second alternative, which we employed in Section 5.4.2, is to perform the parameter space sampling and execution of runs outside ModEx and to collect the results in a directory structure compatible with the above.

### 5.3.3 ModEx app

To explore and analyze the output of the runs, we built the ModEx app as an interactive VisR app. Once users have launched ModEx, they have to first specify the runs’ output directory. The app will then process the directory and show the UI options for further analysis. The ModEx app provides two of the main components of a vPSA framework: computing derived output, and an interactive exploration interface. The following sections describe the design and functionality of each component.

**Derivation**

To be able to effectively analyze the results of many runs of a computer model, we have to summarize its output into objective measures referred to as derived output. However, supporting the functionality to compute the derivations for a wide array of models is an extremely challenging endeavor due to the wide variety of the output types. E.g., image segmentation or clustering algorithms output a different label for each pixel/input element, classifiers output a label for each input, etc. In order to balance between flexibility and usability, our system offers a set of out-of-the-box derivation methods, as well as the means...
to have custom derivation methods when needed. The data flow for the derivation step is illustrated in Fig. 5.2B. Output derivation can be performed in one or more passes. In each derivation pass, shown in Fig. 5.2g, a user selects a derivation method as well as the output data to which the derivation should be applied. The outcome will be a vector of one or more numerical or categorical values per run, examples of which are shown in Fig. 5.2h. These values will be collected and appended to the runsInfo table as new columns with labels generated based on the derivation method as shown in Fig. 5.2i. The choices for out-of-the-box derivations methods are:

- **Aggregate**: includes eight derivation methods. A group of these functions (mean, sum, median, min, and max) are used to aggregate numerical data columns. Another group are meant for categorical output: **Number of class labels** counts the number of different class labels, **Mode of class labels** finds the mode (label occurring the most), and **Count per class label** counts the occurrence (histogram) of each different class label.

- **Comparison with ground truth**: is useful for analyzing models where a data set with ground truth output exists. For example in a supervised classification model, the training data will have ground truth labels. This method will compare the predicted labels for each run with the ground truth labels, and output the number of matches and mismatches per label. A detailed example usage will be discussed in Section 5.4.3.

- **Dimensionality Reduction**: provides MDS, PCA, and tSNE functions. All perform dimensionality reduction on the selected output. An example usage will be discussed in Section 5.4.1.

- **Take first row**: will just use the first row of the selected output table. Despite its simplicity, that is actually what enables embedding custom derivation methods in computer models. It lets a computer model itself to compute any special derived output and export them as a table with a single row. For instance in Section 5.4.1, the clustering app computes clustering quality metrics as derived output for each run.

- **None**: skips the derivation pass altogether. This is helpful when runsInfo contains the necessary derived output already, e.g., when a different tool was used for parameter sampling and derivation, or when the derivation step had already been performed.

We will further discuss the usage of the derivation methods in our case studies in Section 5.4. But for now let’s go through a simple example based on the data flow in Fig. 5.2B. The result of applying the “mean” method on the column “A1” of “TableA” generates a column “mean_A1”. In a second derivation pass, applying the “Count labels” method on the column “A2” of “TableA”, results in two new columns labeled “n(A2=x)” and “n(A2=y)”, corresponding to the total number of “x” or “y” values in “A2” for each run. In another
derivation pass, using the “Take first row” method on “TableB” will take the first rows as-is and add the columns “B1”, “B2”, and “B3” to the runsInfo table.

Fig. 5.3b shows the UI for the derivation step. To keep the user interface simple, each pass will only perform one derivation method on one of the output data tables. However, multiple passes can be performed to generate additional types of derived output as needed. When all passes are performed, a user can select the "Start Exploration" option so that the exploration interface is launched immediately. It is worth noting that the user interface and functionality of the derivation component is also developed as an R-app, so like all other R-apps, its UI and functionality can be easily extended or customized for future needs.

**Exploration**

The core exploration interface of ModEx, shown in Fig. 5.3c-f, allows the user to interactively explore the results of parameter space sampling. Here we will briefly introduce different views and features and will demonstrate more details through the case studies in Section 5.4.

The “Runs overview” in Fig. 5.3d shows an interactive scatter plot of the runsInfo table where each data point is one of the runs of the computer model. As a user hovers the mouse pointer over a run data point, a thumbnail of the output of the run is shown as a tooltip. When user clicks on a run data point, it is selected and its image output is displayed on the “Output of Selected Run(s)” pane as shown in Fig. 5.3e. Multiple runs can be selected by holding the shift button. The user can specify a parameter or derived output column as the scatter plot axis by clicking on the axis label, and can change the visual properties such as the point size, color map, axis scaling mode, etc. using the parameters pane (not shown in the figure). The runsInfo table as well as the output tables for the currently selected run can be accessed through the data pane Fig. 5.3c and are updated as the user changes the currently selected run. This allows the user to chain the outputs to any of the apps available in VisR. We will see examples of this in Section 5.4.1.

The “Runs Distribution” view shown in Fig. 5.3f displays a list of the parameters and derived outputs of the computer model. Each item in the list consists of a histogram (bar plot) and a smaller scale of the “Runs Overview” scatter plot. The histograms have active brushing which can be used to filter the runs with the desired values for the parameters or derived outputs. A user can click on the (-) icons beside each list item, to minimize (collapse) the plots for that list item. The values for the currently selected run(s) and the hovered run under the mouse cursor are indicated by up-side-down triangles above the histograms. The scatter plots use the same axis as the "Runs Overview", but color the run data points based on the values of the parameter or derived output for the corresponding list item to allow a user to quickly check for any specific patterns. Selecting a list item updates the "Run Overview" plot to use the selected item for colouring the run data points. To reduce redundancy we chose to hide the legend by default and instead use the colors shown for
the histogram labels. A user can however turn it back on through the scatter plot app’s parameters.

The core interface described above provides the initial means for exploratory analysis of the runs, however the main strength of ModEx comes from the ability to add other apps to the workflow to extend and customize its functionality based on the specific application domain. We will further explore this in the upcoming section.

5.4 Evaluation

To demonstrate the effectiveness of our framework on different application domains, we will go through three usage scenarios. First, we will look at a clustering problem and compare our results with a tool developed recently to address this problem. Second, we will study the problem of setting hyperparameters for training of neural networks. We will use an educational neural network application as our model and show how ModEx can help find proper values for the hyperparameters. In the third case study, we will look at two commonly used bioinformatics methods and report the insights gained from utilizing ModEx.

5.4.1 Case study: clustering

In our first case study, we will look at the data mining application domain and evaluate the effectiveness of ModEx by comparing it to a recently published visual parameter space analysis tool called Clustervision [315]. Clustervision was developed to help with finding the right parameters for unsupervised clustering. As one of their case studies, Kwon et al. analyze the “Bob Ross Paintings” dataset created by Hickey [339]. They state that they “use this dissatisfaction by Hickey to motivate [their] discussion of how Clustervision could potentially be used to arrive at more satisfactory clusterings”. The “Bob Ross Paintings” dataset hereinafter referred to as the paintings dataset is a dataset about the 403 paintings produced on the PBS show “The Joy of Painting”. It includes 67 binary values per painting specifying the existence of features such as trees, water, mountains, and weather elements, as well as meta information about each painting such as number of used colors and the TV episode name.

Preparations

For this analysis workflow, we developed and added the following R-apps to the VisR framework. It is worth noting that by taking advantage of the available functionality in R for analyses such as clustering and dimensionality reduction, and using VisR’s app development API, developing these R-apps was done in a matter of hours.

- The Clustering app, implements four clustering methods: k-means, spectral clustering [340], hierarchical clustering and DBSCAN [341]. A user can select the input data
Figure 5.3: An overview of ModEx used for visual parameter exploration in a clustering application domain: (a) ModEx parameter view mode for the clustering app; (b) Calculating derived output; (c-f) The generic parameter space exploration interface; (g-i) Additional apps linked to the generic exploration interface to create a customized workflow; (c) Data view providing access to the data tables in the workspace; (d) Runs Overview showing a scatter plot of derived outputs (here the aggregated quality metric vs. number of clusters) for runs; (e) Image Output of Selected Runs highlighted in the Runs Overview plot. (f) Runs Distribution showing a list of histogram distributions and filters for input parameters and derived output; (g) Radar plot showing the quality metrics for the selected runs; (h) Scatter plot linked to the output of the latest selected run; (i) Lineplots / parallel coordinates linked to the output of the latest selected run.

columns to which the clustering should be applied. The app will output two data tables: A table of cluster IDs which has one column and as many rows as the input data table, and a data table of clustering quality metrics computed using the NbClust R package [342]. It also plots a static scatter plot and bounding hull of the dimensionality reduced tSNE projection of the input columns, colored by the cluster IDs. Examples of such output plots are shown in Fig. 5.3e.

- The DimRed app, includes dimensionality reduction methods such as PCA, MDS, and tSNE. It allows users to select several numerical columns on a data table and computes a 2D projection using the specified method.

- The RadarChart app, plots a Radar Chart of user specified numerical columns, an example of which is shown in Fig. 5.3g. This was made to replicate the visual encoding of clustering quality metrics in ClusterVision as shown in Fig. 5.4A.
Prior to starting the parameter space analysis, we imported the paintings dataset in VisR and used the DimRed R-app to compute two dimensional projections of the 67 features. We ran the R-app using the three available dimensionality reduction methods, resulting in 6 new columns to be added to our original input table (two dimensions for each method). This was done in order to better visualize the clustering results in the exploration step.

**Parameter sampling**

To perform parameter sampling on the Clustering app, we first drag-and-dropped it into the VisR workspace and drag-and-dropped the paintings dataset into its view. We then switched the parameter view to ModEx view mode. Fig. 5.3a shows the ModEx view mode and the UI controls to specify the desired parameter ranges for each clustering method. To be consistent with the Clustervision case study, we selected the three clustering methods: k-means, hierarchical, and spectral clustering. For all methods we set the range of the “k” parameter (number of clusters) to \([2, 19]\) and enabled all options for their categorical parameters “algorithm”, “method”, and “kernel”. We finally specified the directory to store the results, set the number of runs to 1000 and started the parameter sampling process. Running clustering for 1000 random parameter combinations took about 2 hours on a 2012 Macbook Pro. Once done, the output directory contained the following:

- The paramsInfo table with the names and types of the parameters.
- The runsInfo table with one row for each run (1000 rows total) and one column per parameter specifying the value assigned to the parameter for each run.
- Two data tables (cluster_ids and quality_criteria) per run (total of \(2 \times 1000\) data tables)
- One output image per run (total of 1000 images).

**Derivation**

The next step after parameter sampling was computing the derived outputs. We added the ModEx app to the workspace and specified the directory of runs to the output directory created in the previous step. For this case study, we ran two passes of the derivation step: In the first pass we chose the derivation method “Number of class labels” on cluster_ids table. This added a new column Num(cluster_ids) to the runsInfo table, specifying the number of clusters generated for each run. Note that even though the number of clusters was specified as an input parameter, it was possible for some clustering methods to converge to fewer number of clusters. In the second pass, we chose the derivation method “Take first row” and specified the quality_criteria table as shown in Fig. 5.3b. We picked the five clustering quality metrics used in the Clustervision including Calinski-Harabaz index, Silhouette Coefficient, Davies-Bouldin index, Gap Statistic and SDbw as well as the mean.
normal value as an aggregated quality metric. This added one column per quality metric to the runsInfo table.

Exploration

In this section we will compare the exploratory analysis of the paintings dataset using ModEx and Clustervision. The main purpose of this case study is to demonstrate how custom parameter space analysis workflows such as Clustervision can be prototyped and developed in our general purpose framework. So we will not be focusing on the effectiveness of Clustervision’s proposed workflow and rather will discuss how much of it could be recreated and possibly enhanced in our framework.

The ranked list view of Clustervision (Fig. 5.4A) shows different clustering results. A set of horizontal colored stripes show a representative of a clustering where the length of the colored stripes represents the number of data points (e.g., paintings) in a cluster. The User can adjust a range slider to focus on clustering sizes relevant to their analysis. On the right of each bar, a radar chart is shown consisting of a sequence of five spokes, each representing one of the quality metrics. The length of each spoke from the center is proportional to the normalized score of the quality metric and they are connected to form a polygon shape.
The initial configuration of the exploration interface of ModEx after start up, contains the views in Fig. 5.3c-f. The Data view in Fig. 5.3c is populated with three data tables. The input table is the input data set, i.e. the meta data of the paintings. The runsInfo table is the parameter values as well as the derived outputs for each run. The Run Output table contains the results for the currently selected run (i.e. cluster IDs) concatenated to the input table.

The “Runs Overview” in Fig. 5.3d takes the runsInfo table as input. We were not able to create the exact user interface of the ranked list view of the Clustervision in ModEx without creating a new app. However to reproduce a similar functionality, we set the x-axis of the “Runs Overview” to the mean normalized metric and the y-axis to the number of clusters. A quick browsing revealed that the clusterings with the higher quality metrics are those with lower number of clusters. This was consistent with the findings in the Clustervision. Alternatively a user could also set the x-axis to any of the metrics to study the clusters based on that metric. This is shown in Fig. 5.5. A user could click to select the run(s) with the desired quality metric and number of clusters. The image output of the selected runs would show in the “Output of Selected Run(s)” (Fig. 5.3e). Also the Run Output data table will update to the output of the main selected run.

Users can also use the sliders in the “Runs Distribution” view (Fig. 5.3f) to filter the runs to a smaller group, for example to focus on specific range of cluster numbers or specific clustering methods. We will further demonstrate the use of this feature in the next section.

To reproduce the radar charts in the ranked list of Clustervision, we drag-and-dropped the RadarChart R-app to our workspace and assigned the runsInfo table as its input as shown in (Fig. 5.3g). Within the RadarChart app’s parameter view, we selected the five columns of quality metrics as the dimensions to be shown on the plot and enabled the “Auto Run” option of the R-app. Every time one or more runs were selected in the “Runs Overview” the RadarChart would refresh to show the polygons formed by the quality metrics of the selected run(s).

The Projection view of Clustervision shows input data points (here the paintings) in a two dimensional tSNE projection, resembling a scatter plot, as shown in Fig. 5.4B. When
users select a clustering result from the Ranked List of Clustering Results view (Fig. 5.4A), the data points in the Projection view are colored to match its cluster.

To recreate this behaviour in ModEx, we drag-and-dropped the Scatter Plot app of the VisR framework to the workspace and assigned the “Runs Output” table as its input. As shown in Fig. 5.3h, we then assigned the cluster ID column as the point colors and the tSNE1 and tSNE2 columns to the x and y-axis (we could also use the MDS or PCA columns). Every time the user was selecting a run in the “Runs Overview”, the Run Output table would be updated and as a result the scatter plot would refresh to draw the points colored by the new cluster IDs for the selected run. In addition we also found it useful to set the image column to the tooltip of the scatter plot so it would show a thumbnail of the paintings as the user moved over the data points.

The Parallel Trends view of Clustervision shown in Fig. 5.4C uses vertical axes to represent each feature of the data points. It draws a line per cluster to show the mean values for each cluster for the corresponding data feature and an area path to represent standard deviation or 95% confidence intervals. We used the existing Parallel Coordinates app of the VisR framework to recreate a similar visual encoding as shown in Fig. 5.3i. Similar to the tSNE scatter plot, we assigned the Run Output table to the input of the Parallel Coordinates app. We then set it up to only show the 67 columns corresponding to the painting features and to use the cluster ID column for coloring the lines. The Parallel Coordinates app, offers an aggregation mode which allows drawing aggregate values such as the mean, median, standard deviation and quartiles for each group of data based on the values in the column specified for the color. We used this mode to show the mean values of the data features for each cluster and specified the column sorting through clicking on the legend (clusterID).

The Clustervision has the option to rank input features in order of importance based on the ANOVA F-Value. The existing Parallel Coordinates app in the VisR framework did not include this feature. Although it was technically possible to develop this new feature in the app, we did not invest on doing this. However, just by sorting features based on the aggregated value of each cluster, we could already detect similar patterns reported by Clustervision.

Fig. 5.6a, shows the top 8 most important features in Clustervision. Fig. 5.6b and Fig. 5.6c show the top 8 features sorted by the aggregated value of cluster 4 (purple) and cluster 2 (red) respectively. We can see that the top features of the purple cluster (ID: 4) of Fig. 5.6b include “Ocean”, “Waves” and “Beach” which are also contained in the green cluster of Fig. 5.6a. Also the red cluster (ID: 2) of Fig. 5.6c includes “Mountain” and “Snowy Mountain” features which are also contained in the yellow cluster of Fig. 5.6a. Additionally we see that “Tree” and “Trees” features are prominent in all but one of the clusters, the green cluster of Fig. 5.6a and the purple cluster of Fig. 5.6c.
Further analysis

While exploring the runs in the “Runs overview” of Fig. 5.3d, we noticed a fairly large group of runs (in red color) that had a much lower value of the mean quality metric regardless of the number of clusters. To further analyze these runs, we used the filters in the “Runs Distribution” view of Fig. 5.3f. The steps of our analysis are shown in Fig. 5.7. First, as shown in Fig. 5.7a, we noticed that all the low quality clusters were using the “Hierarchical” clustering method (red color). So we used the corresponding histogram filter to select those runs, as shown in Fig. 5.7b. We then browsed the other parameters for the remaining runs and noticed a distinct pattern for the “agglomeration method”. As shown in Fig. 5.7c and Fig. 5.7d the values of “single”, “median”, and “centroid” (yellow, orange, and blue colors), were used in the low quality runs. These runs could then easily be removed by filtering out those values.

5.4.2 Case study: neural network playground

Neural networks have gained an unprecedented popularity in recent years thanks to their effectiveness in many application domains. In the context of machine learning, the term
Figure 5.7: Visual analysis of the clusterings: (a) Observing the group of runs with low quality are from hierarchical clustering method; (b) filtering selecting runs using the hierarchical clustering method; (c,d) Observing the group of runs with low quality use “single”, “median” and “centroid” values for the “agglomeration method” parameter.

“parameter” is referred to model variables (such as weights and biases) values of which are learned automatically through a training process. For the constant values that are specified manually to fine tune a model the term “hyperparameters” is used. To solve problems with neural-networks, a machine learning expert would choose a network architecture and set the hyperparameters. These choices are based on heuristics and the expert’s knowledge of the specific problem, and often involve a trial-and-error strategy.

The Neural Network Playground [343] (hereinafter referred to as the Playground) is a web-based interactive application that captures the essence of this task by allowing users to create simple neural networks, visualize the learning progress, and modify hyperparameters. In this section we demonstrate the ModEx workflow and system features using a problem inspired by the Playground application.

Fig. 5.8d shows an example view of the Playground application. The Playground offers two problem types: A binary classification problem and a regression problem. Here we will be focusing on the binary classification problem as it is the more complex one. To generate the dataset for the classification problem, a user selects from one of 4 dataset shapes (Gaussian, Circle, Xor, and Spiral) and a noise value between 0 and 50 and the Playground generates a random two dimensional dataset. The resulting dataset has 200 points which are divided equally into two class labels, -1 and +1, visualized with orange and blue respectively.

To build a neural network, a user can add between 0 to 6 hidden layers (excluding the input and output layers) and set the number of neurons in each layer between 1 to 8 and a fully connected network architecture will be constructed. For the input layer, a user can independently toggle any of the seven input features $X_1$, $X_2$, $X_1^2$, $X_2^2$, $X_1X_2$, $\sin(X_1)$, and $\sin(X_2)$. Before training, several hyperparameters can be specified: “Ratio of training to test data”, “Batch size”, “Learning rate”, “Activation” function, “Regularization” method, and “Regularization rate”. Once done, a user can start the training and the Playground will visualize the results in real time after each training epoch.
Parameter sampling

We were interested in using ModEx to explore the parameter space for this application. The Playground is developed in TypeScript using D3 so creating a VisR app directly from the code was not practical. Instead we decided to reimplement the playgrounds machine learning model using Python and TensorFlow [344] to be able to run the tool in batch mode with different combinations of parameters. At the time of the implementation, we were not aware of the existence of a recent R implementation of TensorFlow so the parameter space sampling is not done in ModEx, but the results were stored in the same structure described in Section 5.3.2. We made an effort to make our implementation as close as possible to the online demo, however, there were occasionally some subtle differences between the outcomes of the two which we believe were the result of the differences between the implementations of the simpler online version of the neural network library compared to the standard Python version as well as the differences in the random number generators. As input, we created one of each of the four dataset types with a noise level of 25. Each input dataset has 200 data points, each with values for the seven input features in the Playground application, as well as its “true” label. For each dataset, we ran the network with 2,500 random combinations of parameters and recorded the results at 4 epochs: 50, 100, 200 and 400. So at the end, for each input dataset, we had a total of 10,000 records in the runsInfo table. For each run we saved an output table of the predicted labels as well as a 2D scatter plot image of the
input data points and the predictions, examples of which are shown in Fig. 5.8c. The points are coloured based on their true label (orange for -1 and blue for +1) and their inclusion in the training or the test set is indicated with a white or black stroke color respectively. The background color of the plot shows what the network has predicted for a particular area after each run.

**Derivation**

The neural network playground script collects several measures after running each configuration of the network for the specified number of epochs. Those include “total time” (training), “mean time”, “train loss”, “test loss”, “train TPR” (True Positive Rate), “train FPR” (False Positive Rate), “test TPR”, and “test FPR”. We could derive some of these measures (e.g. TPR and FPR) within ModEx, but others could only be collected during the training of the network. As such, this example demonstrates how ModEx can still be utilized for parameter exploration of applications developed outside the VisR framework.

**Exploration**

We will now discuss some of the insights from the parameter space exploration of the neural network outputs. We will specifically focus on addressing the feedback from the users of the Playground application stated by Ha [345]: “People started experimenting with different neural network configurations, such as how many neural network layers are actually needed to fit a certain data set, or what initial features should be used for another data set. Which activation functions work better for which dataset?”.

We started by exploring the scatter plots of train loss vs. test loss (Fig. 5.8a), with runs colored by different hyperparameters (Fig. 5.8b). We selected runs that best fit each dataset (i.e. runs that led to low train loss and test loss) by clicking on the points on the bottom left of the scatter plot (Fig. 5.8c). The values of the hyperparameters for the selected runs would also be indicated by a pointing down triangle above the histogram for each hyperparameter. We could also link the workflow to a simple custom R-app that launches the Playground with the hyperparameter configuration of the selected run in the URL (Fig. 5.8d).

Even though this workflow helped us get some initial intuitions, it was still difficult to get any global insights as we didn’t see any obvious patterns in most of the scatter plots. This was partially due to over plotting but also since the effectiveness of the runs was affected by a combination of a large number of hyperparameters rather than a few individual ones. We were interested in studying the hyperparameters for the runs which converged and correctly fit the classes in each data (i.e. runs with small loss). Example outputs of such runs are shown in Fig. 5.9a. So we used the filters shown in Fig. 5.9b to filter out the runs with a high training loss (> 0.2). The percentage of the remaining effective runs were highest for the “gauss” dataset (24%) and lowest for the hardest “spiral” dataset (1%) while the “circle” and “xor” both had around 19% runs remaining after filtering. This wasn’t surprising given
the relative difficulty of the classification for each dataset. Next we set the x-axis to test_loss and browsed through different hyperparameters for the y-axis. Looking through the input features we could see distinctive patterns for each dataset. The input features that had the most effect on converged runs were $X_1$ and $X_2$ for the “gauss”, $X_1^2$ and $X_2^2$ for “circle”, $X_1X_2$ for the “xor” and $\sin(X_1)$ and $\sin(X_2)$ for the “spiral” datasets. Examples of each case are shown in Fig. 5.9c.

We then looked at the effect of the number of layers. As shown in Fig. 5.9d, for the “spiral” dataset a minimum of two layers were required, however the runs for other datasets could converge even without any hidden layers. We could even notice that there were slightly more converged runs when the number of layers were low. This was because these datasets could be fitted easily without any hidden layer given the convenient set of input features. As the number of layers increased, more epochs were needed to converge the network thus runs for lower epochs didn’t pass the filter we set earlier on train_loss.

We next looked at the activation functions. There were four choices for the activation function to be applied to the output of all hidden layers: “linear”, “relu”, “sigmoid”, and “tanh”. The linear activation function, was simply passing the input to output (i.e. $f(x) = x$), while the other three added some nonlinearity to the output of layers, something which is required when classes cannot be fitted properly using only a linear combination of input features. Fig. 5.9e shows the histogram of the runs using each activation function. As expected, all four activation functions performed similarly on the linearly separable gauss dataset. For the “spiral” dataset we saw that only the runs using either of the three nonlinear activation functions converged. However it appeared as if the choice of activation function had little effect on the outcome of the runs on the “circle” and “xor” datasets. Our hypothesis was that those runs were benefiting from the nonlinearity of their input features. So, to verify that, we filtered out all runs which used any input feature other than $X_1$ and $X_2$. As shown in Fig. 5.9f, for the gauss dataset all four activation functions still had a similar outcome, but for the other three datasets, only the runs with nonlinear activation functions converged.

In addition to the patterns we observed for the runs for each specific dataset, we also observed some patterns common in runs for all datasets. As shown in Fig. 5.10a the most effective values for the learning rate seemed to be in the [0.001, 1.0] range. To study regularization rate, we first selected runs that used either L1 or L2 regularization (Fig. 5.10b) and observed that the most effective values for the regularization rate seemed to be in the [0.001, 0.3] range. We were also interested in the computational cost of the networks. We explored association of different hyperparameters with the average epoch time and noticed the batch size and neuron count to be the two hyperparameters showing the strongest association with the mean epoch time (Fig. 5.10d).
Figure 5.9: Selected plots from exploration of runs for the Playground case study: (a) Example outputs from converged runs for each dataset; (b) Filtering out runs with high train_loss; (c) Example of input features showing high correlation with converged (low loss) runs; (d) Exploring effect of the number of hidden layers; (e) Exploring the effect of activation functions; (f) Exploring the effect of activation functions on runs that only used $X_1$ and $X_2$ as input features. Except for the gauss data, a nonlinear activation function is needed to train a successful classifier.
5.4.3 Case Study: differential gene expression

In this section we demonstrate the effectiveness of ModEx using a case study in differential expression analysis performed in collaboration with a group of biologists.

Differential gene expression (DGE or DE) analysis is one of the common analysis in genome biology and is used to identify genes (or other genomic features) that are expressed in significantly different quantities in distinct groups of samples [346]. These data are acquired using genome sequencing technologies (e.g. RNA-seq) from samples collected in different biological conditions such as drug-treated vs. controls, diseased vs. healthy, different tissues or different stages of development. Bioinformaticians have developed several computational methods such as Limma [334], edgeR [160], DESeq [291] and DESeq2 [161] that employ various statistical models to identify differentially expressed (DE) genes. These methods take two groups of gene expression datasets (with two or more replicates per group) as input and assign a label of either “0”, “-1”, or “+1” to each gene. A none DE gene is labelled “0”, while a DE gene is labelled “+1” when it is detected to be expressed in significantly higher quantity in the second group (a.k.a. up-regulated) or “-1” when it is the other way around (a.k.a. down-regulated).

Our users were interested in using DESeq and edgeR as the two most commonly used methods for DE analysis [299]. R-apps for these methods were already incorporated in VisR as shown in Fig. 5.11a and Fig. 5.11c. One of the main questions that biologists had when using these apps, was whether the genes were classified as DE through a stringent computer model (i.e. specificity) or more tolerant one (i.e. sensitivity). They also wanted to have control over the sensitivity. That was because sometimes in earlier stages of their studies they wanted to cast a wide net to identify a large set of possible candidates and later perform more rigorous focused analysis (e.g. pathway analysis or wet lab analysis) to narrow down the results, while other times they wanted to be most confident about the detected DE genes.
Parameter sampling

In order to study the effect of parameters on the sensitivity of DE analysis methods, we used a benchmark gene expression dataset called SEQC (a.k.a. GSE49712) [347]. The data was prepared under two biological conditions, and for each condition, the sample collection and measurement was repeated 5 times (biological replicates). In addition to the genetic material from tissues, a mixture of 92 synthetic genes, called spike-in, was added to each sample with known abundance, so the DE states of those genes were known in advance. For 25% (23) of the spike-in genes the ratio of genetic material added to both groups of samples was identical, so they were expected to be detected as none-differentially expressed. For the remaining 75% (69) spike-in genes the ratio was different (0.5, 0.67 or 2) so they were expected to be detected as differentially expressed. The final sequenced and processes dataset was stored in a data table with 21716 rows (one per gene, 92 of which were spike-in), 10 columns of gene expression values (5 for each of the two biological conditions) and one column “GT_DE”, indicating the ground truth DE state, with a value of TRUE or FALSE for the 92 spike-in genes, and NA for the remaining of the genes. First we used the ModEx view for each app shown in Fig. 5.11b and Fig. 5.11d and performed the DE analysis for
500 runs with random combinations of input parameters. The output of each run was the computationally determined DE state for each gene. This was stored as a data table with one row per gene, and two columns, “DE” with a value in {-1, 0, +1} and “isDE” with a value of FALSE when DE=0 or TRUE when DE=-1 or +1.

**Derivation**

Our goal was to compare the actual DE state of the control spike-in genes, with the DE state computed by the DESeq and edgeR apps given a set of parameters. We studied each of these apps separately. For derivation, we used the “Comparison: with ground truth” derivation method as shown in Fig. 5.11e and set it to use the “isDE” column of the app’s outputs with the “GT_DE” column of the input dataset. Each gene’s computed DE condition, “isDE”, could take one of the two values of TRUE or FALSE and the ground truth DE condition, “GT_DE”, could take one of the three values of TRUE, FALSE, or NA, so there were a total of six combinations possible for the output of each run for each gene:

- isDE=TRUE & GT_DE=TRUE: indicating a true positive
- isDE=TRUE & GT_DE=FALSE: indicating a false positive (type I error)
- isDE=FALSE & GT_DE=FALSE: indicating a true negative
- isDE=FALSE & GT_DE=TRUE: indicating a false negative (type II error)
- isDE=TRUE & GT_DE=NA: possibly a DE gene
- isDE=FALSE & GT_DE=NA: possibly a none DE gene

The derivation step counted the combinations for each run. This resulted in 6 columns that were added to the runsInfo table. Each column contained the count of genes for one of the 6 combinations. For example, the column “n(isDE=TRUE & GT_DE=TRUE)” contained the number of true positives for each input parameter combination.

**Exploration**

We will now discuss our findings using the ModEx exploration interface, first for the DESeq app and then for the edgeR app. For studying the sensitivity of a classifier model, the “receiver operating characteristic” plot (ROC plot) is one of the most well known visualizations. The ROC plot is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various parameter settings [348].

To get a ROC plot, we selected the “n(isDE=TRUE & GT_DE=FALSE)” (to represent the FPR) as the x-axis and the “n(isDE=TRUE & GT_DE=TRUE)” (to represent the TPR) as the y-axis of the plots. Note that these represented counts (not rates) as they
were computed by the derivation method without the knowledge of the total counts. To compensate for this, we set the x-axis maximum to 23 (the number of none DE spike-in genes) and the y-axis maximum to 69 (the number of DE spike-in genes).

Fig. 5.12a shows one of the ROC plots for DESeq runs. Each point represents one run (i.e. an input parameter combination) and the color represents the value used for the categorical parameter “method”. In a ROC analysis, the optimum points are those for which there is no other point with both lower FPR and higher TPR. These points form a set known as the Pareto set and are referred to as the Pareto frontier in a ROC plot. Browsing through the ROC plots, we observed that a large group of points in the Pareto set were colored red as shown in Fig. 5.12a. Those were the runs that used the value “blind” for the “method” parameter, so we used the histogram filter Fig. 5.12b to select them. We then observed that the categorical parameter “sharing mode” was determining the Pareto set, more specifically the two values of “fit-only” (red) and “maximum” (green), as shown in Fig. 5.12c. So we used the histogram filter once more Fig. 5.12d to select runs using those two values. The remaining points were those either in the Pareto set or very close to it. For these points, the parameter “FDR-threshold” had a clear correlation as shown in Fig. 5.12e. Runs with a lower value for “FDR threshold” had a low FPR and TPR, while those with a higher value for “FDR threshold” had also high FPR and TPR. A value of near 0.4 seemed to be a reasonable value to have keep the FPR low at 4% (\(\frac{1}{23}\)) and TPR high at 87% (\(\frac{69}{699}\)). Last but not least, a forth parameter “fit type” seemed to have a slight effect on the number of detected DE genes. As shown in Fig. 5.12f, when plotting the total number of DE genes vs. the FDR threshold, we could see that a value of “local” resulted in detecting slightly more DE genes compared to a value of “parametric”.

We performed a similar ROC analysis on the runs of the edgeR app. Unlike DESeq, we initially didn’t see a single parameter that would clearly define the Pareto set. As shown in
Figure 5.13: ModEx exploration of the edgeR app. ROC plot of the runs with the x-axis set to FPR and the y-axis set to TPR and colored by the parameters: (a) the “CPM cutoff”, (b) “P-value adjustment method”, and (c) “Method for differential test”. (d) effect of “P-value adjustment method” (x-coord) and “Method for differential test” (color) parameters on the FPR (y-coord). (e) Selecting runs using “glm likelihood ratio test” value. (f) Positive correlation between “P-value” (color) parameter and FPR (y-coord). (g) Effect of “CPM cutoff” (x-coord) and “P-value adjustment method” (color) on the number of detected DE genes (y-coord). (h) Effect of “CPM cutoff” (x-coord) and “P-value” (color) on the number of detected DE genes (y-coord). (i) “Method for differential test” parameter biases the detection of down-regulated (x: n(DE=-1)) and up-regulated (y: n(DE=1)) genes.

Fig. 5.13a The “CPM cutoff” parameter had a negative correlation with the TPR, but didn’t directly effect the FPR. On the other hand, the “P-value adjust method” parameter had some effect on the FPR but not directly on TPR: As shown in Fig. 5.13b the value “holm” (purple) resulted in runs with lower FPR. The “Method for differential test” parameter was another parameter that had some effect on the FPR: As shown in Fig. 5.13c, runs with “glm likelihood ratio test” parameter value (blue) resulted in relatively lower FPR than the runs with “exact test” value (red). To further explore the parameters affecting FPR, we set the plots to show FPR vs. “P-value adjustment method”. The plot that used “Method for differential test” parameter as the point color, had a visible association. As shown in
Fig. 5.13d the “holm” value for “P-value adjustment method” parameter (x-coordinate) resulted in overall reduced FPR among other groups, but within each group, the runs using “glm likelihood ratio test” parameter value (blue) had always less FPR than those using the “exact test” parameter value (red). Note that there is a slight jitter on point coordinates to handle over-plotting. So we used the histogram filters (Fig. 5.13e) to select the runs with “glm likelihood ratio test”. This made it easy to observe that the “P-value” method had a positive correlation with FPR as shown in Fig. 5.13f.

In order to explore the effect of parameters on the total number of detected DE genes, we set the plots to show the number of detected DE genes, “n(isDE=TRUE)”, vs. the “CPM cutoff” parameter. As shown in Fig. 5.13g and Fig. 5.13h, the number of detected DE genes decreases by increasing the value of “CPM cutoff”. The “holm” value for “P-value adjustment method” parameter detected the least number of DE genes (Fig. 5.13g). For each selected value of the “P-value adjustment method” parameter the “P-value” parameter had a slight positive correlation with the number of detected DE genes (Fig. 5.13i).

We also noticed that the “Method for differential test” parameter introduced a bias in detecting up-regulated (DE=1) vs. down regulated (DE=-1) genes. As shown in (Fig. 5.13h), when using “exact test” value (red), more up-regulated (DE=1) genes were detected. In contrast, using the “glm likelihood ratio test” (blue) showed a bias toward detecting more down-regulated (DE=-1) genes.

To summarize, we found that the DE results of the DESeq method seem to be closer to an ideal ROC, than the edgeR method. For DESeq it seemed the parameter combination of {'method': 'blind', 'sharing mode': 'fit-only'} should be used and the 'FDR threshold' can be used to control the specificity of the detection method.

5.5 Conclusion and future work

In this chapter we discussed our novel general purpose system for visual parameter analysis of computer models. By offering key components of a visual parameter analysis framework including parameter sampling, deriving output summaries and an exploration interface, as well as a flexible API for further extension, ModEx can be employed in a variety of application domains with a reduced development time. We demonstrated the usability and flexibility of our system in three application domains: data mining, machine learning and bioinformatics. However there remains several important limitations that provide opportunities for extending the current work. We would like to address these limitations while we continue evaluating our system on more scenarios in different application domains.

5.5.1 Parameter sampling

Our current implementation only uses random uniform sampling of the parameter space. We did not experiment with other sampling methods as it was not a direct focus of our
study. We leave detailed studies of different parameter sampling methods, such as grid sampling, or stratified random sampling, as important future work. Moreover, ModEx does not include the “prediction” component of the conceptual framework for visual parameter space analysis the purpose of which is to predict or estimate model outputs for parameter combinations that have not been sampled.

5.5.2 Analyzing output

Currently we only allow a side-by-side comparison of output images in the exploration interface. However, we believe better comparison methods of images as well as output data tables will improve the effectiveness of analysis. A different type of useful comparison, is the comparison of multiple computer models together. Currently to compare different models together, a user needs to first combine them into a meta-app similar to our approach for the Clustering app. Another missing feature is the ability to group runs with similar output (images or data) to reduce the clutter during exploration and better help with finding patterns or anomalies. Note that in some cases a brute-force grouping by comparing the output values may not work well, for example when the output are IDs for cluster labels.

5.5.3 Browsing parameters

Our current implementation displays parameters and derived output independent of each other. However in many models, dependent parameters may exist. For example in our Clustering app, the algorithm parameter is only relevant when the clustering method is set to k-means. The R-app API already allows declaring these dependencies to show parameters only as needed, however this information is not utilized during the exploration. We initially experimented with a hierarchical view of parameters instead of a list, however the resulting UI turned out to be too complex and we opted out to a basic list view until we find a better alternative. Another useful feature when browsing parameters is to be able to sort parameters by some sort of importance metric that measures the significance of the parameters on the output of the models.

5.5.4 Guiding the exploration

In our current exploration interface the user has to manually experiment with different configurations of axis to look for interesting patterns. One area of potential improvement is to utilize scagnostics [349] to help the user find interesting views of the data.
Chapter 6

Conclusions

In this thesis we presented two main scientific contributions:

- a novel general purpose framework that bridges the gap between the biologists and the bioinformaticians through a system of visual analysis modules that can be rapidly developed and connected together.

- a first-of-its-kind system for enabling visual parameter analysis framework of a wide variety of computer models.

VisR provides: (a) tools for analysis of sequencing data, (b) a framework for rapid development of R modules called R-apps, (c) support for interactive exploration of the results, (d) support for linking apps to create more complex workflows, and (e) an ecosystem to allow extension and sharing of the apps. ModEx extends VisR to offer a general visual parameter space analysis framework that provides: (a) parameter sampling, (b) deriving output summaries, and (c) interactive and customizable exploration interface.

We started by presenting an overview of data and task abstractions for the analysis of epigenomics data and justified design and implementation of the ChAsE tool. ChAsE is published as a design study paper [350] and an application paper [196] and is available to download from the website: http://chase.cs.univie.ac.at.

We then presented VisR, a general framework for analyzing sequencing data and creating interfaces for R libraries. We demonstrated that by reducing the required technical expertise, VisR facilitates data analysis for a broader set of biologists and bioinformaticians. VisR, is published as a systems paper [168] and the software package and documentations are available from the website: https://visrsoftware.github.io.

Finally, we presented ModEx, a general purpose system for exploring parameters of a variety of computer models. We discussed how ModEx offered key components of parameter analysis frameworks and explained how it can be used for rapid development of custom parameter space exploration solutions for different application domains.
ChAsE and VisR have been listed in the websites of the International Human Epigenome Consortium (IHEC)\(^1\) and the Canadian Epigenetics, Environment and Health Research Consortium (CEEHRC) Network\(^2\) (Fig. 6.1). We also had the opportunity to introduce VisR to the Bioconductor community during the annual BioC conference\(^3\). The materials for the workshop titled “Making R packages accessible to non-programmers using the VisRseq framework” are available at: https://github.com/hyounesy/bioc2016.visrseq

Figure 6.1: VisR featured at the front page of the website of the Canadian Epigenetics, Environment and Health Research Consortium (CEEHRC) Network.

### 6.1 Pitfalls of visual analysis systems

Even though visualization offers a powerful aid to understanding biological data by combining the computational power with the human beings’ cognitive ability in pattern recognition and interpretation, we should recognize that visualization systems will not be of benefit to every biological question.

In cases where the analysis tasks are well-defined and the regularities in the data are clearly understood it is often possible to develop effective algorithmic solutions that address the challenges solely by computation. An example of this is finding peaks in epigenetic datasets where algorithmic solutions can effectively separate the location of epigenetic modifications from background noise. However, even in those cases, visualization may still

\(^1\)http://ihc-epigenomes.org/research/tools

\(^2\)http://www.epigenomes.ca/tools-and-software

\(^3\)http://bioconductor.org/help/course-materials/2016/BioC2016/
offer complementary benefits, for instance in querying and finding patterns regarding the presence of the peaks as we discussed with the ChAsE tool.

Another concern with making interactive visualization tools readily accessible to a broader community is that such tools can either lead, or be misused by, uninformed users toward incorrect insights and findings, for instance by facilitating cherry picking of correlations that are consistent with the users’ hypothesis. To address this, not only the research communities need to be better educated on the principles of data analysis and statistics as well as their limitations, the tool designers must carefully choose appropriate techniques based on the analysis tasks and the goals of the visualization.

6.2 Future directions

We already reflected on the lessons learned for each of our main contributions and provided detailed future directions in Section 3.8, Section 4.5, and Section 5.5. We conclude our thesis with a discussion of the limitations of VisR where there is a potential for future work.

6.2.1 Provenance and reproducibility

While a primary goal of exploratory visual analysis tools is to aid the users with gaining insights and eventually lead them to discoveries, the information about the process is itself of great value and needs to be well documented to be able to reproduce the results and communicate the findings. So an important step in scientific research specially in biological sciences is the collection and representation of data provenance information such as the history of changes and advances throughout the analysis process. Failure in doing so can endanger the reproducibility of the findings. Of the three systems described in this thesis, only ChAsE had some limited support for recording provenance information by allowing the users to label intermediate results and to save them as favorites. However VisR and ModEx would both substantially benefit from a system that not only automatically keeps a history of the analysis and exploration provenance information, but also allows the users to annotate the process. A good start would be to investigate the recent works on provenance systems for visual analysis such as CLUE [351], AVOCADO [352], and the Refinery platform [353].

6.2.2 Extending to other application domains

Our primary focus during the development of VisR was on supporting the analysis of sequencing data. As it currently stands, VisR is used in gene expression analyses (i.e. RNA-seq data), epigenomics analyses (e.g. ChIP-seq and DNA methylation data), and single-cell analysis. Among future directions we believe VisR should be adapted to be employed in other biological studies specially those with a strong presence in the R and Bioconductor
community\textsuperscript{4} including proteomics, flow cytometry, mass spectrometry, immuno oncology, and pharmacogenomics.

A natural next step would be extending VisR as well as ModEx toward a general visual analysis package for a broad range of scientific studies. This will require collaboration with scientists from different disciplines as well as developing a new version with a web-interface and a back-end with support for cloud-computing. Our vision is to eventually encourage the R package developers to also create companion R-apps for their packages themselves and make them available through a central app repository. For that, we need to continue including a rich collection of apps to the framework ourselves. Even though most developers would like to make their packages accessible to a broader range of users, the platform must first be adopted by a large group of scientists.

\textsuperscript{4}http://bioconductor.org/packages/release/BiocViews.html#___ResearchField
Bibliography


