Variant View

Visualizing Sequence Variants in their Gene Context

by

Joel A. Ferstay

B.Sc., The University of British Columbia, 2006

B.Sc., The University of British Columbia, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2013

© Joel A. Ferstay 2013

Abstract

Scientists use DNA sequence differences between an individual’s genome and a standard reference genome to study the genetic basis of disease. Such differences are called sequence variants, and determining their impact in the cell is difficult because it requires reasoning about both the type and location of the variant across several levels of biological context. In this design study, we worked with four analysts to design a visualization tool supporting variant impact assessment for three different tasks. We contribute data and task abstractions for the problem of variant impact assessment, and the carefully justified design and implementation of the Variant View tool. Variant View features an information-dense visual encoding that provides maximal information at the overview level, in contrast to the extensive navigation required by currently-prevalent genome browsers. We provide initial evidence that the tool simplified and accelerated workflows for these three tasks through three case studies. Finally, we reflect on the lessons learned in creating and refining data and task abstractions that allow for concise overviews of sprawling information spaces that can reduce or remove the need for the memory-intensive use of navigation.

Preface

This thesis is based on material contained in the following paper

- Joel A. Ferstay, Cydney B. Nielsen, and Tamara Munzner. Variant View: Visualizing Sequence Variants in their Gene Context. *To appear in Transactions on Visualization and Computer Graphics (Proc. InfoVis 2013)*.

Table of Contents

Abstract	ii
Preface	iii
Table of Contents	iv
List of Tables	vii
List of Figures	viii
Acknowledgements	x
Dedication	xii
1 Introduction	1
1.1 Approach and Contributions	2
1.2 Thesis Organization	5
2 Variant Analysis Pipeline	6
2.1 Next Generation Sequencing	6
2.2 Sequence Variant Data Generation	8
2.2.1 Quality Control	8
2.2.2 Alignment/Mapping	9
2.2.3 Filtering Variant Candidates	9
2.3 Prioritization of Candidate Variants	11
3 Visualization Design Process	12
3.1 Nine-Stage Design	12

Table of Contents

4	Data and Tasks	15
4.1	Data	15
4.1.1	The Reference Genome	15
4.1.2	Scales and Coordinate Systems	16
4.1.3	Variant Attributes	18
4.1.4	Gene Attributes	18
4.2	Tasks	20
4.2.1	Driving Biological Tasks	20
4.2.2	Tasks and Data Questions	22
5	Related Work	24
5.1	Genome Browsers	24
5.1.1	Sequence Read Visualization	25
5.1.2	Genome Browsers for Variant Analysis	27
5.2	Tailored Gene View Solutions for Variant Analysis	28
6	Design Rationale	32
6.1	Core Components: Automation versus Visualization	32
6.2	Genomic Coordinates: Strengths and Weaknesses	35
6.3	Filtered Scope	36
6.4	Transcript and Protein Region Encoding	40
6.5	Variant and Variant Attribute Encoding	41
6.6	Use of Color	42
6.7	Design Comparison	43
6.8	Implementation	44
7	Case Studies	47
7.1	Case Study 1: Discover	47
7.1.1	Hypothesis Confirmation	48
7.1.2	Hypothesis Generation	49
7.2	Case Study 3: Compare	50
7.3	Case Study 4: Debug Pipeline	51

Table of Contents

8 Discussion, Future Work, and Conclusions	53
8.1 Specialize First, Generalize Later	53
8.2 Visualization Design Considerations	54
8.3 Design Progression	55
8.4 Design Study Methodology Pitfalls Analysis	62
8.4.1 Pitfalls in the Winnow Stage	62
8.4.2 Pitfalls in the Cast Stage	66
8.4.3 Pitfalls in the Discover Stage	67
8.4.4 Pitfalls in the Design Stage	70
8.5 Scalability Limitations	70
8.6 Future Work	71
8.7 Conclusions	74
Bibliography	76

List of Tables

4.1	Data abstraction table.	19
4.2	Data question table.	23

List of Figures

1.1	A genetic sequence variant.	2
1.2	Variant View tool screen capture.	4
2.1	The variant data generation and analysis pipeline.	7
3.1	The Nine-Stage Design process.	13
4.1	Biological context in which variants occur.	16
5.1	The Ensembl genome browser	26
5.2	The IGV [43] sequence read visualization tool.	27
5.3	The Ensembl [5] variation image.	30
5.4	The cBio [4] tool for analysing genetic variants.	31
5.5	The MuSiC [9] tool.	31
6.1	Variant View tool.	34
6.2	A detail view of the top of an Ensembl [5] variation image.	39
6.3	Variant visual encoding.	45
6.4	Comparison of the same variant data.	46
7.1	Confirming AML genes.	49
7.2	Discovering AML genes.	51
7.3	Comparison of patient data.	52
7.4	Debugging the bioinformatics pipeline.	52
8.1	The first two data sketch prototypes.	56
8.2	An interactive search supporting prototype.	57
8.3	A second interactive search supporting prototype.	58

List of Figures

8.4	An intermediate prototype for the Variant View design	59
8.5	A sixth data sketch prototype with icons for variant type. . .	60
8.6	A prototype showing protein symbols and amino acid classes.	61
8.7	Design Study Methodology: 32 pitfalls	63
8.8	A hierarchical Aggregation Example.	73

Acknowledgements

I would first like to thank my thesis supervisors, Dr. Tamara Munzner and Dr. Cydney B. Nielsen. Working with Tamara was a tremendously valuable experience. Tamara’s knowledge of the information visualization literature and vast experience in effectively presenting complicated topics in a lucid fashion helped make my work much more accessible and understandable to both a visualization audience and a genome sciences audience. Cydney was a fantastic co-supervisor. Cydney not only helped to connect me with potential collaborators at the Genome Sciences Centre, where I performed a great deal of this project, but also provided immeasurable support and understanding in the face of potential projects that could have become sprawling, large team efforts. Cydney gave me both guidance and the confidence to step up to these projects and neatly carve out a well-scoped project that was not only interesting, but could also be completed within the confines of a masters program.

I would also like to thank Dr. Karon MacLean, the second reader for this thesis. I have reaped the benefit of taking several of Karon’s courses at the University of British Columbia, and have always found her courses to be stimulating, and immediately practical. I feel Karon’s expertise and thoughtful comments on this thesis have made it much more accessible to a broader audience, even to those outside the domain of information visualization and the domain of genome science.

I would like to thank the members of the InfoVis group, Matthew Brehmer, Jessica Dawson, Stephen Ingram, and Michael Sedlmair, for their insightful and very constructive feedback on my work. I would also like to thank them for the opportunity to comment and provide feedback on their work, which was such a pleasure.

Acknowledgements

I would further like to thank Rod Docking, Dr. Gerben Duns, Dr. Linda Chang, Simon Chan, and Dr. Aly Karsan, our collaborators at the British Columbia Cancer Agency and Genome Sciences Centre, for their patience, time, and interest in this project.

I would also like to thank Jessica Dawson, Anna Flagg, Shathel Haddad, Juliette Link and Louise Oram for being excellent support during this process. It was always a pleasant diversion to ask about your projects, be inspired by your creativity, and then return, energized, to my own work. I would also like to thank Meghan Allen for being an inspiring teacher; I am hopeful her teaching skills rubbed off on me: they are probably the reason for my ability to achieve a teaching assistant award.

I would like to thank the Vancouver Institute for Visual Analytics (VIVA), MITACS, and AeroInfo/Boeing for their generous funding support.

Finally, I cannot begin to describe all the support my family has given to me. My parents Ralph and Cheryl Ferstay, my brother Daniel Ferstay, and Emanuel (Manny) Cabral: their love and support make each day a new one.

Dedication

To my parents, Ralph and Cheryl, who have given me so much. To my grandparents, Alfonse and Ivy, who I miss.

Chapter 1

Introduction

The human genome project produced a reference genome for the human species [15], consisting of about 3 billion chemical constituents called nucleotides. Each person’s genome is slightly different; the rate of variation between the nucleotide sequences for individuals is less than roughly one percent [23]. Differences between an individual person’s genome and the reference genome are called *sequence variants*; Figure 1.1 shows an example sequence variant. Changes at the DNA sequence level can cause a variety of genetic diseases such as cancer. Scientists are interested in finding sequence variants that are predictive of different disease states, and they do so by comparing the genome sequences of individuals diagnosed with a disease to the reference genome, which is generally assumed to be healthy and disease-free. This problem is non-trivial because very few variants are harmful and teasing these apart from the much larger set of harmless variants requires both automated detection and human inspection. Human reasoning about the biological impact of variants is particularly challenging because it requires considering multiple attributes at a variant position across several levels of biological context.

Currently, variant analysts attack the problem with workflows that have high cognitive load because of the need to mentally integrate across many databases and spreadsheets. The dominant visualization tools for exploring sequence data in general are genome browsers [5, 11, 16, 43, 47]; using them typically requires extensive navigation with very high time costs. A few systems have been proposed for variant analysis, but they either share the fundamental problems of genome browsers [5] or fall short of presenting the full spectrum of biological context needed by the analysts [4, 9].

In this design study, we worked with four variant analysts over a six

Figure 1.1: A genetic sequence variant. A person’s genome is compared to a reference genome for the human species to identify where and how their genome varies in nucleotide content. In the above example, the person has a variant G nucleotide at position 6.

month period to design and refine Variant View, a tool to accelerate and improve variant analysis, shown in Figure 1.2. We identified three variant analysis tasks: finding candidate genes that may be implicated in specific types of cancer, comparing data about an individual patient to a data set of variants known to be harmful, and debugging the bioinformatics pipeline before the data is used for any further analysis.

1.1 Approach and Contributions

One contribution of this thesis is a data and task abstraction for the problem domain of variant analysis: our task analysis links concrete, domain-specific questions to this data abstraction. Another contribution is a discussion that reflects on the strengths and weaknesses of genomic coordinates as a data abstraction, a question that has broad implications for the design of biological visualizations. A third contribution is the validated design and implementation of Variant View. We carefully justify our choices for visual encoding and interaction techniques with respect to the data and task abstractions. With careful filtering, we created an information-dense overview for multiple, non-contiguous features at multiple scales showing all necessary information simultaneously without the need to navigate. We validate the effectiveness of the tool with three case studies of its use after several months of deployment. Our final contribution is a discussion of the

1.1. Approach and Contributions

lessons learned in this design study: the design strategy of “specialize first, generalize later”, and six design considerations organized into the themes of “what to show” and “how to show it”.

Figure 1.2: Variant View tool screen capture. Sequence variants and their attributes shown in Variant View with respect to biological context annotations at multiple scales. This gene, whose name is anonymized, was not previously known to be implicated in leukemia; analysts identified it as a candidate gene through variant analysis using the tool.

1.2 Thesis Organization

Chapter 2 describes the variant analysis pipeline from start to finish at a high level to give context to the problem addressed by Variant View.

Chapter 3 describes the methodology used in this design study for finding collaborators and a promising visualization problem. We apply the design study methodology of Sedlmair et al.[38] to create a visualization solution for our collaborators. Chapter 3 also describes the process for translating this problem into a refined set of data and task descriptions that guide design and implementation of a visualization tool.

Chapter 4 describes the data types and attributes used by analysts and the driving biological tasks for the problem of variant analysis; Chapter 4 is central to defining requirements for the system.

Chapter 5 presents related work. The aim of this chapter is to describe current visualization approaches to the problem of variant analysis, the data attributes important to the task that they reveal, their strengths, and why they are insufficient solutions for the current analysis task.

Chapter 6 outlines a principled approach to the design of a display and visual encoding for the problem of variant analysis. In this chapter we leverage existing knowledge of human perceptual and cognitive strengths and weaknesses to design an information dense visual encoding for variant analysis. The choice of what data or attributes to include and emphasize or de-emphasize in this visual encoding is prioritized by the discussion in Chapter 4.

Chapter 7 provides three case studies that provide initial evidence for our claim that the Variant View tool helped analysts perform their work faster, and see patterns in the data that they may not have seen without it.

Finally, Chapter 8 reflects on the lessons learned during this design study, and possible design criteria and guidance for future studies.

Chapter 2

Variant Analysis Pipeline

This chapter describes the variant analysis pipeline from start to finish at a high level to give context to the problem addressed by Variant View. The contents of this chapter summarize information mainly drawn from a review of current variant research methods by Altmann et al. [1], and a current survey of variant research methods by Pabinger et al. [33].

Variant analysis requires knowledge of how variant sequence data is produced. This knowledge is important because obtaining variant data involves a multi-stage pipeline of operations, and artifacts in the form of spurious variants can be introduced at several stages of the pipeline. The data production pipeline involves producing variant data of sufficiently high quality based on various metrics. After data production, analysis takes place. This analysis usually requires knowledge of biology to mentally integrate several pieces of information to determine whether a variant is harmful or not [1, 33]. Figure 2.1 depicts this process from end to end. This chapter outlines the complete process at a very high level to motivate problems in this domain, some of which are solved by tools discussed in the Related Work of Chapter 5, and expose a new problem, whose solution is the focus of this thesis.

2.1 Next Generation Sequencing

Analysts interested in sequence variant analysis must first acquire the data. The first step is obtaining a whole genome sequence from an individual. Although ideally one would like to acquire a long, continuous, error-free, sequence of nucleotides representing the entire genome, current DNA sequencing technology is incapable of reading the entire genome sequence continuously from one end to another. The current technologies used to perform

2.1. Next Generation Sequencing

Figure 2.1: The variant data generation and analysis pipeline. This pipeline begins with Next Generation Sequencing (NGS), where nucleotide identities are reported. A quality control step removes nucleotides after the NGS process that are unlikely to be correct based on various score metrics. The alignment and mapping step aligns nucleotide sequence data to the reference genome to produce a list of sequence variants. The number of variants at this stage can be large, so the filtering variant candidates step involves application of automated algorithms to remove variants that are unlikely to be of interest. Prioritization of variants takes place after filtering; this step requires an analyst with biological knowledge to inspect the list of variants to determine which variants are predictive of disease or interesting for further research. Variant View is designed to support analysts at the prioritization of variants step. The last step is Validation; after analysts are certain that a particular list of variants warrant further research, a technique such as Sanger sequencing is used to validate or confirm that these variants are true.

this are a number of *platforms* collectively called next generation sequencing (NGS). A platform is the sequencing technology and the associated software for translating raw chemical signals identifying nucleotides into text files of nucleotide sequence reads. NGS allows for sequencing of the whole genome

of single individuals in a single laboratory within two weeks and at a low cost compared to the earlier Sanger sequencing method [1, 33, 37]. NGS involves extracting DNA from a cell population. The pipeline of today's most widely applied sequencing NGS platforms entails the fragmentation of the DNA to be sequenced into smaller segments called sequencing *reads* [1, 33]. The output of a NGS experiment is a collection of millions of these short reads. Each NGS platform introduces sequencing errors that are characteristic for its sequencing pipeline. Compared to traditional Sanger sequencing, these high-throughput sequencing approaches produce many more sequences, but of much shorter length and inferior quality; the shorter length of reads and inferior quality has an impact on how the resulting readouts are processed in a downstream analysis [1, 33, 37].

The technology of NGS is subject to an ongoing development and the current generation of sequencing technologies are about to be replaced by more modern approaches aimed at eliminating some of the current technical problems which result in lower quality data. However, even if new NGS technology produces more error-free data, this data will still need to be analyzed.

2.2 Sequence Variant Data Generation

Obtaining genetic variant data requires a sequence of steps following generation of the NGS data. We discuss those steps that are most likely to lead to erroneous data in the final sequence data due to uncertainty in their process. The steps discussed are Quality Control, Alignment/Mapping, and Filtering Variant Candidates.

2.2.1 Quality Control

Most sequencing platforms provide the DNA sequence read data directly in a flat file format. Checking the quality of the generated sequence data is the first step in the pipeline that deals with the actual sequence data. NGS platforms produce quality scores for each individual nucleotide in a

read, and some of them automatically remove data if it does not meet the platform manufacturer’s factory-specified threshold [1, 33].

2.2.2 Alignment/Mapping

For almost all applications, sequence reads are aligned to a reference sequence; in this case, the human genome. This requirement for aligning several million short reads, which contain small deviations and sequencing errors, to a reference sequence or a database of sequences has brought forth a number of efficient algorithms some of which use hashing to accelerate the alignment step. The choice of alignment tool and the corresponding settings significantly affect the outcome and may cause errors to appear in downstream processing. The choice of alignment algorithm and its parameter settings are both important. One such parameter setting is closeness of match between sequence reads and the reference. For instance, if only perfect matches between sequence reads and a reference are allowed, the downstream analysis will not find any differences between the reference and the sequenced genome, and no variants will be found. On the other hand, allowing many mismatches between the reference and sequence reads may allow for many wrong alignments and result in a high number of false positive variants in the downstream analysis [1, 33].

Once the reads have been aligned to the reference genome, many algorithms allow storage of the result in the sequence alignment/map (SAM) format [1, 33]. The SAM format stores information about each aligned read. At this point, visual inspection of a whole genome sequencing experiment is usually not realistic. However, one can isolate alignments within a target region and visualize only that specific region in a genome browser such as the Integrative Genomics Viewer (IGV) [43].

2.2.3 Filtering Variant Candidates

Filtering is an essential step to reduce the number of false positive variant *calls*: a call is the presence and nucleotide identity of the DNA sequence made by the sequencing technology [1, 33]. Algorithms designed to remove

2.2. Sequence Variant Data Generation

these false positive variants and minimize variant calling artifacts are GATK, SAMtools, and VCFtools [8, 19, 21]. Most variant calling tools have the option to generate the data in the VCF format [8].

After this round of filtering, and only once the reads are aligned, variants can also be filtered out if they do not have sufficient read support: for instance, if variants are only supported by only a single read or very few [1, 33].

Variants can also be filtered out based on the effect they might have on the cell. To facilitate this process, the VCF format specifies for each variant basic information such as the chromosomal position, the reference nucleotide, and the variant nucleotide. Information on the quality of the variant call as well as the amount of sequence data available for the call are stored. The variant calling process on whole genome data can generate more than a million variants. To cope with this size, tools for automated variant annotation have been developed. Effect predictor algorithms such as snpEff [6] can enrich the variant data in the VCF file with information relating to the gene the variant takes place in, and the other possible effects the variant could have on the cell downstream. Variants that are predicted to have very low impact on the cell may be removed from the data at this point [1, 33].

A widely used approach to substantially reduce the candidate list is to exclude known variants which are present in public variant databases, published studies or in-house databases as it is assumed that common variants represent harmless variations. The entire filtering process helps reduce a variant data set from millions of variants, to thousands [1, 33].

Working with NGS systems is an interdisciplinary effort. While the generation of the data is mainly laboratory-centered, the initial processing of the short read data falls into the domain of bioinformatics and is mostly automated. The interpretation of the results, however, requires close interaction between biology and bioinformatics in order to derive insights from the data; there is a need for better tools at this interface [33].

2.3 Prioritization of Candidate Variants

With the use of whole-genome sequencing, the challenge is narrowing down the list of candidate variants and interpreting remaining variants within a biological context. Prioritizing the resulting filtered variants is task specific; it typically requires detailed knowledge about the domain and visual inspection of the variant context. After an analyst is confident that their list of variants are mostly true positives, and biologically interesting, they verify the list of variants using Sanger sequencing [1, 33, 37]. This list of variants can vary in size depending on the task; for our project, analysts worked with lists of between 2,000 and 10,000 variants. They might spend up to 15 minutes analyzing each variant.

Our visualization solution, Variant View, fits into this pipeline at the prioritization of candidate variants stage where analysts must use their knowledge of disease biology and the nuanced information from the effect predictor algorithms to determine which variants to report for validation methods such as Sanger sequencing. Figure 2.1 shows where Variant View is situated in the variant analysis pipeline. As we will discuss in detail in Chapter 5, there is a lack of visualization tools to guide this prioritization process. The next chapter will demonstrate the design process we adopted to create our visualization solution.

Chapter 3

Visualization Design Process

Our design process followed the collaborative nine-stage design study methodology framework of Sedlmair et al. [38] because it is the culmination of three advanced visualization researchers’ experience in co-authoring over twenty design studies, and their literature survey of many more. Sedlmair et al.’s [38] design study methodology framework targets the design of visualization tools for complex data sets. We applied this existing design framework to our design process in this project.

3.1 Nine-Stage Design

The nine stages are the precondition phase of *learn*, *winnow*, *cast*; the core phase of *discover*, *design*, *implement*, *deploy*; and the analysis phase of *reflect*, *write*. This process is depicted in Figure 3.1. In this study, the three visualization researchers were new, moderately experienced, and very experienced; given this combination of expertise, we did not allocate time for an explicit learning phase. We did indeed have an extensive winnowing stage of roughly five months, in which we considered several other biological problems of potential collaborators at the Michael Smith Genome Sciences Centre (GSC) but decided against pursuing them. We discuss this process in more detail in Section 8.4. We ultimately selected the problem of variant analysis as a rich problem domain with interesting visualization research questions after a series of meetings with two front-line analysts (A1 and A2) who are research biologists. We made connections with these two postdocs through a gatekeeper (G1) who is engaged in both basic and clinical research at the GSC.

The core phase of the design study lasted roughly six months. During

3.1. Nine-Stage Design

Figure 3.1: The Nine-Stage Design process advocated by Sedlmair et. al. [38]. Figure courtesy of Tamara Munzner.

this time we met with analysts regularly, for around an hour a week, and their feedback and ideas actively shaped the tool capabilities. The discover stage began with several semi-structured interviews with analysts A1 and A2 to understand their current workflow and identify tasks that visualization might address. Their tasks are described in Section 4.2: their main problem is the Discover Genes task. The design and implementation stages were tightly interwoven, with a series of 8 prototypes of increasing complexity created over five months. We decided that data sketches [20] were more appropriate than paper prototyping due to the complexity of the data, so even the earliest prototypes did load and show real data. The first two prototypes were static tests of visual encoding possibilities, where we received feedback by demonstrating them to the analysts. The deploy stage began in the third month with the third prototype, which supported interactive search; from then on, A1 and A2 used the prototypes in their analysis process, with each new prototype replacing the previous one. Five more prototypes of increasing sophistication were deployed over the next two months, and A1 and A2 have been using the final version for two more months. A1 and A2 used the tool whenever they needed to assess variant impact, and reported that it helped them to see patterns in their data that were difficult to imagine using previous tools such as their spreadsheet software. We comment more

on A1 and A2's experiences with the tool in Chapter 7.

When this prototype was demonstrated to gatekeeper G1, he became enthusiastic about using it for other biological problems. He connected us with two more analysts, A3 and A4, who are bioinformaticians. Their feedback identified the driving problems described as the Compare Patient Task and the Debug Pipeline Task in Section 4.2. Based on analyst feedback, we adapted the base design to handle these additional tasks with two more rounds of prototyping over one month. These analysts were intrigued by this prototype and are considering how it might be incorporated into future workflows. Deployment for the Debug task with A3 and A4 might be possible in the near future, since they have direct control over their own workflow. However, deployment for the Compare task is a more complex problem since that workflow is still being developed and gatekeeper approval is required for clinical use. A staged development process, as with LiveRAC [22], would be one way to approach the problem; we leave it as future work.

The analysis phase of the design study overlapped with the core phase, and extended for another month beyond it. As usual, the writing stage triggered a return to the discover stage to further refine the data and task abstractions, which in turn led to a few improvements in design. Writing also triggered a return to the reflect stage, as we considered what lessons we learned that might be of interest to visualization practitioners who have no connection to this particular domain.

Chapter 4

Data and Tasks

Before we present the visual encoding and interaction design choices of Variant View, we need to explain the underlying data and task abstractions [26, 29]. We begin with the data abstraction, where we explain the characteristics of the domain data and how we abstract it in terms of scale and type, and discuss the computation of derived data. We then explain the tasks in more detail with respect to data involved, and then consider what abstraction in domain-independent language is interesting. The data and tasks were extracted through continued feedback from our analyst collaborators as part of our design process.

4.1 Data

This section outlines the variant data attributes and data capturing important levels of biological context that are necessary for variant analysis.

4.1.1 The Reference Genome

A variant is a difference between an individual person’s genome and the reference genome, a standardized coordinate system derived as a consensus from a small number of people. Because the reference genome is assembled from several donor genomes, comparison of a given genome to the reference will expose millions of genetic variants. Many of these variants are just due to harmless genetic differences between this sample genome and the reference. The reference genome is an imperfect abstraction, and is actively being augmented by sampling [42] and storing [12, 39] the larger scope of human variability. The genome of an individual is called a *sample*; it typically con-

Figure 4.1: Biological context in which variants occur. “X” symbols indicated sequence variants between a sample genome from an individual person and the reference genome. Variants propagate from the genome level to the transcript and protein levels and can be harmful (red) or harmless (blue) depending on whether they disrupt important biological signals or not. The gene level is specified by genomic coordinates, the exon-containing transcript level is specified by transcript coordinates, and the protein level is specified by protein coordinates.

tains thousands to millions of variants. Our collaborating analysts working with cancer genomes applied several rounds of custom filtering to identify on the order of hundreds of variants of interest per individual. When summed across roughly a hundred samples, their data sets contain between 2,000 and 10,000 variants.

The starting point of variant analysis is of course the variants themselves, but they need to be interpreted within a larger biological context of additional information about the genome and its structure. Figure 4.1 shows a diagram of the relevant biological context.

4.1.2 Scales and Coordinate Systems

As with many complex datasets, there is known and relevant structure at multiple scales in genome sequence data (Table 4.1). At the top level is the entire genome, which is roughly 3 billion nucleotides (nt) in length. The

reference genome establishes *genomic coordinates*, a linear coordinate system that specifies location within the sequence as an nt index. The standard way to provide information about known biological context is as *annotations* that pertain to a range between two locations.

The next relevant level of structure below the genome itself is *genes*; they are roughly 10,000 nt in length, and there are approximately 20,000 genes in the human genome. Below the gene scale, the next level is *exons*, the part of the gene sequence that creates proteins. They are roughly 100 nt in length and there are on average 10 of them per gene.

Eliminating the regions of the genome that are not exons leads to a second coordinate system, *transcript coordinates*. Exon ordering is preserved between *genomic* and *transcript* coordinates, and these regions are simply stitched together to produce transcripts that are on average 1000 nt long. Most genes in the human genome produce multiple different transcripts. This diversity results from a biological process of different subsets of exons being assembled into alternative transcripts under different conditions. For example, one transcript may include all of a gene's exons while another may include all but one exon.

Finally, each triple of nucleotides in the transcript is translated into one amino acid to create a sequence that is a third as long, for a third coordinate system of *protein coordinates* indexed by amino acid (AA).

The lowest level of relevant structure is *protein regions*, which range from one to hundreds of AA in length and are specified in protein coordinates. These regions have known functional properties, such as facilitating chemical reactions within a cell or anchoring the protein to particular structures. There are around 20 types of protein regions; each has a list of ranges in protein coordinates specifying known regions of that type. Proteins do not typically have annotations for all possible region types, but rather only have annotations for a few region types. Variants that cause amino acid changes within these regions are considered more likely to disrupt protein shape or function.

4.1.3 Variant Attributes

Each variant can have a position in all three coordinate systems: genome, transcript, and protein. Each variant has several categorical attributes, also summarized in Table 4.1 in terms of coordinate system and the number of categories for each. There are 4 possible nucleotide types for a variant, represented by the well-known letters A, C, G, or T, and there are 20 possible AA types, classified into 4 different chemical categories. A variant that changes the AA to one of a different class is typically more disruptive than those where the new AA is still in the same class. There are 7 possible variant types, for example, a nt insertion or a nt deletion. Another attribute of interest is whether a variant is recorded in any of the two major databases that categorize certain variants as known to be harmful or known to be harmless. These databases are imperfectly curated and cover multiple cancer subtypes, so this information is considered supplemental rather than definitive. Each variant also has an associated list of sample identifiers; the same variant may occur in only one or multiple samples.

4.1.4 Gene Attributes

Our data abstraction also includes two derived attributes, called *varcount* and *hotspot*, that we calculate for every gene (Table 4.1). These attributes were not previously used by the analysts, but capture patterns that we determined were of interest based on our task analysis. The *varcount* metric is simply the count of how many variants occur within a gene normalized by the gene’s length in nt, and thus it ranges from 0 to 1. This measure is useful for identifying highly mutated genes. The *hotspot* metric is a more complex metric that goes beyond counts to capture the co-location of variants within a gene. We group together neighboring variants if their distance is smaller than 20 nt in transcript coordinates; this threshold corresponds to the inflection point in the distribution of inter-neighbor distances for all genes. The *hotspot* metric is then computed as the maximum group size for a gene, and thus it ranges from 0 to the maximum value for the data set. This metric is useful for identifying genes with large clusters of variants.

4.1. Data

Annotations	CS	Count	Length
Gene	G	20K per genome	10K nt
Exon	G/T	10 per gene	100 nt
Functional Region	P	10 per gene (20 region types)	1-300 AA
Variant Attributes	CS	# Categories	
Variant Position	G/T/P	-	
Nucleotide Type	G/T	4	
Variant Type	G/T/P	7	
Amino Acid Type	P	20	
Amino Acid Class	P	4	
Database Status	-	4	
Sample IDs	-	(list)	
Derived Attributes		Range	
VarCount Metric		[0, max]	
Hotspot Metric		[0.0, 1.0]	

Table 4.1: Data abstraction table. Annotations may be in one or several coordinate systems (CS): Genome (G), Transcript (T), or Protein (P). Their counts are given, and their average lengths show their relative scales. Variant attributes are also associated with a coordinate system. For categorical attributes, the number of categories are shown. Each sample has a unique identifier and two gene-level derived attributes.

4.2 Tasks

All of the analysts were in a group at the GSC focused on the specific cancer type of acute myeloid leukemia (AML). In this section we first characterize the problems and tasks our analysts face and then distill a set of questions they ask about their data to perform these tasks.

4.2.1 Driving Biological Tasks

Discover Genes: The Discover Genes task is to find new genes that are candidates for involvement in the disease of acute myeloid leukemia through variant analysis. The scope of this task is limited to hypothesis generation; the identified candidate genes would then be investigated further to confirm those hypotheses with other tools. This task takes place within the context of an extensive pipeline of data processing and analysis. The input at this stage is a dataset of around 3,000 variants that has already been pre-filtered by data quality metrics. Each variant is associated with several attributes including the gene within which it occurs; typical datasets have around 50 variants per gene, and include samples from around 100 individuals.

The analysts loaded this list into a spreadsheet, sorted by gene name, and then went through line by line to make judgements about the impact of each variant by reading its attributes. They also used web-based tools to determine whether the variant appears within any of a large number of *protein regions*. This latter task required an arduous process of querying a protein database website, selecting a protein from a list of possible proteins, inspecting the resulting web page of protein details, and mentally intersecting the variant’s location within the genome with the interval of the protein region boundaries; this process can take about 15 minutes for each variant. They also manually compared the variant against two different databases of known variants [12, 39], to understand whether or not it had already been characterized as being harmful or harmless; consulting these databases requires about 5-10 minutes per variant.

Compare Patient: The Compare Patient task is to compare variant data

for a particular individual patient with a database of variants that are known to be harmful for acute myeloid leukemia, in hopes of generating a diagnosis and treatment plan by noting variants similar to a disease population group [41]. The known-AML database contains around 10,000 variants with at most 200 variants per gene; the patient dataset typically has around 1000 variants, with at most 10 variants per gene.

The challenge is that *similarity* is loosely understood rather than fully characterized. A specific variant in a patient clearly corresponds to a known one if they have exactly the same position and attributes; the question of whether nearby variants should be considered matches is more fuzzy. Currently, A3 and A4 are in the process of developing algorithms that classify variants into three categories: positive matches, unclear, and unlikely to match. Their preliminary algorithms generate reports that are being used experimentally by clinicians as part of a workflow that is still under development. Although they are not clinicians themselves, A3 and A4 work closely with them, understand the pain points of the current prototype workflow, and have access to real patient data. They conjecture that visualization support might allow the clinicians to better interpret the border cases between matching and non-matching where the algorithm may fall short, and possibly also to better use the matching variants for a treatment plan.

Debug Pipeline: The Debug Pipeline task is to ensure that the bioinformatics pipeline used to generate variant datasets from raw data is working correctly, before relying on the output in downstream tasks such as Discover Genes or Compare Patient. There are several places that errors might occur in the multi-stage pipeline: spurious variants may be generated due to noise in the next-generation sequencing stage or incorrect thresholding in the data quality filtering stage after that, and incorrect attributes for variants might be generated by the variant effect prediction stage. The goal of finding biologically implausible results requires knowledge of both biology and the variant data production pipeline. Once a pattern is known to reliably predict false positive data, it can be incorporated later into automated filtering algorithms. Although the bioinformaticians already had debugged

their pipeline extensively, visualization support has often uncovered errors of a kind difficult to detect with other methods.

4.2.2 Tasks and Data Questions

Table 4.2 contains the full list of concrete questions about the data that we identified for three target tasks of Discover Genes, Compare Patient, and Debug Pipeline.

The Discover Genes task involves Q1 through Q9. Q1 through Q4 are direct questions about variant attributes. The only unimportant attribute is the list of sample IDs; these unique identifiers are occasionally used to look up further information but are not directly of interest themselves. Q5 is about proximity between variants themselves. Q6 and Q7 also pertain to proximity, but specifically whether a variant falls within given annotation ranges. Q1 through Q7 are all at a *gene-level* scale; that is, they only pertain to variants within a single gene. Q8 and Q9 are at a larger scale: they characterize genes with respect to each other in terms of patterns of variants within them. These two questions are at *genome-level* scale; they pertain to selecting which genes to inspect in more detail. The Compare Patient Task involves Q10 and Q11: these questions also pertain to the positions of variants and their attributes. Finally, the Debug Pipeline Task involves Q12 and Q13: Q12 is purely about position, and Q13 is a direct question about variant attributes.

Identifying the questions analysts ask about their data can provide guidance for what information is required to solve their tasks, and what information is irrelevant. In the design rationale discussion of Chapter 6, we use these concrete questions to motivate and justify the design decisions we make to construct our visualization solution.

Discover Genes Task: Gene-Level	
Q1	What is the variant type?
Q2	Is there a change in AA chemical class? From what to what?
Q3	Is there a change in AA? From what to what?
Q4	Is the variant in any of the known databases? Is it a harmless or harmful one? Which one(s) is it?
Q5	Are there many variants in close proximity to each other? Where?
Q6	Is the variant close to an exon boundary?
Q7	Which types of functional regions are known for this gene and does the variant fall within any range of any of them? If so, which types? Which ranges?
Discover Genes Task: Genome-Level	
Q8	Are there genes with many variants?
Q9	Are there genes with variants in close proximity to each other?
Compare Patient Task	
Q10	Does a patient variant occur at exactly the same position as a known variant? If so, do the attributes match exactly (variant type, AA change, nt change)?
Q11	Does a patient variant have a known variant nearby it? If so, are the attributes the same? Or very similar?
Debug Pipeline Task	
Q12	Is there an unusual or biologically implausible distribution of variants?
Q13	Is there an unusual or biologically implausible combination of attributes at a variant position?

Table 4.2: Data question table. Concrete questions asked by analysts to infer variant impact, for each of the three identified tasks.

Chapter 5

Related Work

There are many tools available for visualizing sequence variant data. These tools vary in their approach, with some targeting flexibility in the sense of displaying a large number of attributes, some irrelevant to the current design study’s tasks, to others that are more targeted to particular tasks with limited flexibility. Generally speaking, genome browsers [5, 11, 16, 43, 47] are the most flexible tool, and even allow visualization of raw sequence read alignment to a reference genome [3, 10, 14, 43]; one genome browser in particular, Ensembl [5], allows for a specialized display for variant analysis in addition to typical genome browser capabilities. Other representations expose variant attributes more explicitly, such as cBio [4] and MuSiC [9], but only do so at the gene level.

5.1 Genome Browsers

Genome browsers are the dominant paradigm in sequence visualization today [5, 11, 16, 43, 47]. At their core is the data abstraction of *genomic coordinates*: the genome is considered as a single, long, linear sequence of nucleotides, and nucleotide position within the string acts as an index. The visual encoding is that horizontal spatial position reflects genomic coordinates, with interactive navigation through panning and zooming to adjust the view to show any single region of interest. Multiple rows are stacked vertically into *tracks*; each of these separate tracks can show any kind of data that can be indexed with respect to genomic coordinates. An enormous amount of genomic information is indexed this way in public and private biological databases, as *annotations* that refer to some range in genomic coordinates.

When zoomed all the way in, the user sees features at the level of individual nucleotides, including their actual values as C, G, A, or T in the base track. When zoomed all the way out, the entire genome is shown. Even when zoomed out only to the gene level, individual nucleotides cannot be resolved, and there are many irrelevant regions present which causes regions of potential interest to be so highly compressed that useful information is not visible. Figure 5.1 shows a screen capture of the Ensembl genome browser zoomed in to the gene level. Variant data is added as a track. Tracks can stack vertically. Genome coordinates are shown at the top. Variants are encoded as thin, vertical lines, and color encodes variant type. Stacked below the variants are other data tracks such as transcript information and protein domains. The variant data are so squished that they just appear as thin, non-salient vertical lines. The strengths and weaknesses of genomic coordinates are discussed further in Section 6.2 as part of our design rationale.

5.1.1 Sequence Read Visualization Tools for Variant Analysis

The output of alignment technologies, used to help identify sequence variants and described in Section 2.2.2, is a collection of read alignments of nucleotides A,C,G, and T to a reference genome. If many short reads aligned to a similar region of the genome have the same nucleotide in a particular position, there is greater evidence that the nucleotide is a true positive, and not simply due to some error in the data generation pipeline. Tools such as Artemis [3], Bambino [10], Magic Viewer [14], and IGV [43] support viewing of these aligned reads by displaying them stacked and/or end-to-end. Figure 5.2 depicts the IGV tool, arguably the most popular of these. The short, horizontal, grey tracks in this figure are sequence reads. The reference genome sequence of nucleotides are depicted at the bottom. The vertical column that is highlighted in blue shows all the read evidence for a particular position. Here, all the reads are showing a C, meaning that there is a lot of evidence that the individual’s genome had a C in this position.

5.1. Genome Browsers

Figure 5.1: The Ensembl genome browser. Variant data is added as a track. Tracks can stack vertically. In this screen capture, the genome coordinates are shown at the top. Variant data is shown within the black box we have drawn to emphasize their presence. The thin vertical lines are variants, and color encodes their variant type. Stacked below the variants are other data tracks such as transcript information and protein domains.

The analyst would therefore be likely to trust this C nucleotide assignment based on the stacked read evidence. Generally, these kind of read visualization tools are good for detailed inspection of read evidence for a single variant. However many other attributes that variant analysts are interested in such as how the sequence variant will impact its gene’s protein product is not available; read alignment visualization tools are not tailored for the problem of variant impact analysis as described by the data and tasks we have defined in Chapter 4.

5.1. Genome Browsers

Figure 5.2: The IGV [43] sequence read visualization tool. Two tracks are shown. The grey-filled horizontal strips represent DNA sequence read fragments. Along the horizontal x-axis is a genomic coordinate system. Each grey read fragment is aligned to a reference genome at the bottom. Read fragments from the same experiment are in the same horizontal track. The grey histograms above each track shows read coverage, the number of reads supporting a particular nucleotide, per nucleotide.

5.1.2 Genome Browsers for Variant Analysis

The Ensembl [5] genome browser, in addition to being a fully-fledged genome browser tool, now includes support for visualizing variants and their attributes in the form of a so-called variation image [5]; the variation image is shown in Figure 5.3. The variation image encodes variant type and some variant attributes, in addition to partially collapsing the inter-exon regions to give more screen space to variants within exons. This view shows only a single gene in a display, but it does not provide any guidance on what gene to inspect. One benefit of the single-gene approach is that its scalability problems are less extreme than those of a general-purpose genome browser in terms of panning and zooming to regions of interest in the entire genome. However, the Ensembl variation image’s track-based view, shown in Figure 5.3, typically requires vertical scrolling, particularly to see vari-

ants across multiple alternative transcripts: each possible transcript and its associated protein regions form a unit, and around ten of these units are stacked vertically to span a great deal of screen space. The representation also requires user interaction to expose some attribute information such as known database type, and does not encode AA class. Variants are difficult to resolve since they are encoded as thin vertical lines. Their type is encoded by color, which is difficult to resolve because the variant lines are so thin. Also, because inter-exon regions are only partially condensed, exon regions are still small, and multiple variant lines in close proximity can overlap and occlude each other making it difficult to resolve variant type, position, and recurrence. Finally, at this time, the Ensembl genome browser’s variation image does not allow analysts to upload their own data into the system to be displayed. They can only visualize variant data from existing, curated datasets.

5.2 Tailored Gene View Solutions for Variant Analysis

In contrast to genome browser approaches, there are two recent tools, cBio [4] and MuSiC [9], that are more tailored to the display of variant attribute and multi-scale annotation information; cBio is shown in Figure 5.4 and MuSiC is shown in Figure 5.5.

These tools are a useful first step in showing important feature information at the overview level. Both show variant position with respect to annotation boundaries. However, several visual encoding decisions lead to difficulties in using them to assess variant impact. For example, cBio encodes the repetition of multiple variants as variant bar height, which is only minimally salient. MuSiC encodes repetition of multiple variants at a single location with vertical stacking if they are identical, and triangular bloom-like layouts if they are collocated but of different type. Both MuSiC and cBio are missing much of the detailed information of amino acid class and known database type. In both cBio and MuSiC, protein regions are likely

5.2. *Tailored Gene View Solutions for Variant Analysis*

to overlap, leading to occlusion and difficulties in resolving what regions are affected by variants. Neither tool shows where variants occur in relation to the gene transcript, thus it is difficult to know whether variants occur in and around exon boundaries. A major barrier to cBio use is that users cannot import their own data. Although MuSiC is technically available as open source, it too has barriers to use: the undocumented code to generate plots is embedded within a larger system codebase and would be nontrivial to adapt for standalone use.

5.2. Tailored Gene View Solutions for Variant Analysis

Figure 5.3: The Ensembl [5] variation image's track-based view typically requires vertical scrolling, particularly to see variants across multiple alternative transcripts since each possible transcript and associated protein regions are stacked. The full display is labelled (A); this view extends across nearly seven pages when printed out directly from the browser. Variants are encoded as thin, vertical, colored lines. The region labelled (B) shows a magnified cropping of the display that includes a transcript scope similar to Variant View. The region labelled (C) shows protein regions which are also included in Variant View.

5.2. Tailored Gene View Solutions for Variant Analysis

Figure 5.4: The cBio [4] tool for analysing genetic variants. The tool represents variants as a red circle on a thin vertical line. The color of the circle does not encode any information. Information about protein coordinate and amino acid change is given in text above the variant. The entire protein length is encoded by a grey rectangle, and protein domains are encoded as colored boxes on top of the protein length. Variant count at a single position is encoded by the height of the variant circle and line. Additionally, an amino acid coordinate scale is given at the bottom of the screen.

Figure 5.5: The MuSiC [9] tool for analysing genetic variants. Variants are encoded by a thin vertical line with a circle on top. The color of the circle encodes the variant type. The protein length is encoded by a horizontal rectangular box, with protein regions along this length encoded as colored boxes on top. An amino acid coordinate scale is given along the length of the protein. Amino acid coordinates and identities are given in text above each variant.

Chapter 6

Design Rationale

We now discuss the design decisions for Variant View.

6.1 Core Components: Automation versus Visualization

Figure 6.1 shows Variant View, with its core interface components labeled. The overall design arose from considering the specific tasks outlined in Section 4.2 and identifying three common themes. First, analysts need to integrate diverse data types from distinct sources, such as patient variant data in user-specified input files or protein annotations from public databases. Manually integrating these data together one gene at a time as described in Section 4.2.1 is very time-consuming. We therefore decided to automate this process by building data integration into Variant View so that all relevant data is available from within a single unified interface. Second, analysts need to prioritize genes based on these integrated data, but the previous workflow only provided alphabetical sorting by gene name. We designed two derived metrics, varcount and hotspot, and equipped Variant View with a reorderable list of genes that can be sorted by either of these metrics or alphabetically (Figure 6.1, label B). This component of Variant View also supports direct searching by gene name. Finally, analysts need to make judgements about the biological significance of a gene’s variants. Unlike the other two general tasks described above, this one requires human inspection and we therefore designed a concise visual interface to support this type of reasoning. We strove to encode as many attributes into the primary overview (Figure 6.1, label A) as possible; to avoid clutter, we show

6.1. Core Components: Automation versus Visualization

attributes that were deemed by our analysts to be more peripheral to the analysis into the supporting table view (Figure 6.1, label C). Variant View features bidirectionally linked views [46] such that selections in any one of the views are reflected in the others; the video included in the supplementary materials shows the look and feel of the interaction at more length. The video is available at: http://www.cs.ubc.ca/labs/imager/video/2013/variantview/variantview_video.mov

Figure 6.1: The Variant View tool, annotated to indicate its three main views. The primary view (A) is the central overview for performing variant impact assessment; the reorderable gene list view (B) can sort genes alphabetically or by derived measures of variant importance; the secondary Variant Data table (C) contains peripheral information.

6.2 Genomic Coordinates: Strengths and Weaknesses

Although many genome browsers provide access to many hundreds of public data tracks, an analyst typically focuses on fewer than a dozen tracks at once; only a few tracks are relevant to a specific task at once. The data abstraction of genomic-coordinate tracks provides an extremely flexible architecture, allowing new data types to be easily incorporated into genome browsers. The popularity of genome browsers implies that many tasks in this domain are well served by this style of pan-and-zoom navigation. Users can easily navigate to a known range and explore local neighborhoods around it at that same scale. They can also easily synthesize information about correlation between phenomena in the same range across multiple tracks. The fixed coordinate system allows users to easily preserve and maintain orientation in terms of where some feature of interest lies with respect to larger-scale structures in the genome.

However, genome browsers are more difficult to use for tasks that require understanding features that fall into non-contiguous regions because the interaction costs become high. Extensive panning and zooming adds both time cost and cognitive load for the user, who must remember regions of interest and their context because they cannot be seen side by side [18, 35]. Genome browsers are particularly difficult to use when features of interest have distributions that are sparse or bursty across some range. The problem with sparse distributions in a fixed coordinate system is that the features are small relative to the scale of the range in which they fall, so they are difficult or impossible to see when the user has zoomed out far enough to see the full range. Similarly, distributions with bursts of features very close to each other can be difficult to understand from high zoom levels because they lie on top of each other, so that a burst is hard to distinguish from a single occurrence. Genome browsers are also difficult to use when features of interest fall at multiple scales, so they cannot be easily seen at any single zoom level. Moreover, if an analyst does not already have hypotheses about what regions in a dataset are interesting, it could be difficult to find such

areas through unguided exploration. Abstractly speaking, the problem is a lack of information scent [13, 34] in the overview; that is, at high zoom levels there is no visual indication of what areas might be fruitful to explore next, forcing users to undertake exhaustive search.

Collapsed coordinate systems can be used instead of genomic coordinates to emphasize regions of interest. They are a much less common representation than genomic coordinates. As discussed in Chapter 5, tools like the Ensembl variation image [5] use partially collapsed inter-exon regions to slightly emphasize exons as regions of interest. Overall, collapsed coordinates risk not being able to show data that fall outside of the selected regions, and they also distort the scale, which may be important for some tasks.

The data abstraction of genome coordinates is sufficiently powerful and pervasive that it has widespread use, but variant analysis is one of many biological subdomains where it falls short [30, 31]. In our design we abandoned them completely, in favor of the collapsed coordinate systems of transcript and protein coordinates as a way to filter the scope of what is shown.

6.3 Filtered Scope

A central design decision was to aggressively filter out all information unnecessary for variant analysis tasks in order to create an easily-comprehensible overview showing everything important simultaneously. All questions except for Q8 and Q9 require seeing only a single gene at a time. The analysts ignore all variants that occur outside of gene boundaries both because their functional consequences are much more difficult to assess and because they are deemed less likely to be harmful.

Even Q8 and Q9 do not require visually encoding the location of the genes in genomic coordinates. Thus, there is no overview of the entire genome in the main view; only a single gene is shown at once. The gene to inspect is selected from a reorderable list of gene names in a secondary view that can be sorted according to the derived attributes of hotspot and varcount, to satisfy Q8 and Q9, and reduce the gene search space.

Moreover, considering the information summarized in Tables 4.1 and 4.2 in combination shows there is no need to use genomic coordinates at all; transcript and protein coordinates suffice. That is, our task analysis also shows that there is no need to show the non-exon parts of the gene that do not contribute to the transcript, so we filter them out completely. Again, the analysts ignore all variants that occur outside of exon region boundaries, deeming them unlikely to be harmful.

We also realized that there is no need to show low-level nucleotide or protein type information at non-variant positions. Thus, we only show the boundaries of annotation ranges, without attempting to show their internal structure.

In a traditional genome browser, each sample would be shown separately with its own horizontal band. We have instead chosen to show all of the variants together in the context of a single coordinate system, combining information across all samples. Again, this decision was motivated by our task and data analysis: no question requires direct comparison between multiple samples. The only questions that require reasoning about an individual sample are Q10 and Q11. We once again handle the problem with aggressive filtering: in that case, we only show two more variants for each one in the individual sample, its neighbors to the left and right.

We relegate secondary information to an auxiliary spreadsheet-format table linked to the main view: it contains details about identifiers in known databases (for Q4c, *which database is it?*), the genomic coordinate value, identifiers for samples and the transcript, and other attribute information in textual format that is also visually encoded in the main view.

These decisions lead to a view dramatically different from what is shown in a traditional genome browser: it is information-dense but visually clear, showing all important information simultaneously without the cognitive load of navigation.

Figure 6.2 shows a detail view of the top of an Ensembl [5] variation image, annotated to show the distinction between genomic coordinates and exon regions. A screen capture of the Ensembl variation image is shown in Figure 6.1. Genome coordinates are encoded as the alternating black

6.3. *Filtered Scope*

and white bar at the top. This view includes both the exon regions and regions between them. The exon regions are much smaller than the non-exon regions, so in genomic coordinates that give equal weight to every nucleotide position the exons are so squished that their internal structure is difficult to distinguish. In contrast, the track beneath shows an alternate view where exons are expanded horizontally and the non-exon regions are compressed. Our design takes this idea a step further by eliminating the non-exon regions completely.

Figure 6.2: A detail view of the top of an Ensembl [5] variation image annotated to show the distinction between genomic coordinates and exon regions. A screen capture of the Ensembl variation image is shown in Figure 6.1. The exon regions are much smaller than the non-exon regions, so in genomic coordinates that give equal weight to every nucleotide position the exons are so squished that their internal structure is difficult to distinguish. In contrast, the track beneath shows an alternate view where exons are expanded horizontally and the non-exon regions are compressed.

6.4 Transcript and Protein Region Encoding

With the scope of the view reduced to a focus on the exon-containing transcript and protein regions, the next decision was how to encode them. Using horizontal spatial position for the coordinates was the obvious choice, given the strong precedent of horizontal encoding of coordinate data in genome browsers. The transcript and protein regions are bars on separate vertical rows, as shown in Figures 6.1 and 6.3, with the transcript bar on top showing exon boundaries within it (Q6) and one row for each protein region type below it. They are aligned to have the same spatial extent so that vertical locations correspond across these rows, supporting reasoning simultaneously across these levels. To make this alignment possible, the protein coordinate system is at a higher zoom level than the transcript coordinate system. For instance, if 900 transcript coordinates are captured by the orange transcript bar, 300 protein coordinates will be stretched to match up with the transcript. This representation makes sense because 3 transcript coordinates is equal to 1 protein coordinate, as described in Section 4.1.2. For instance, hypothetical transcript coordinates 4, 5, and 6 will all map to protein coordinate 2. Encoding each type of functional regions as its own bar on a separate vertical row is an important choice to prevent occlusion while accommodating Q7; in previous systems these intervals all fall into the same spatial region, leading to overlap and visual clutter. The transcript bar is the top, orange bar depicted in Figure 6.1 label A. The protein region bars are the bars stacked below the transcript bar in Figure 6.1 label A.

Only the protein region types that appear as annotations for a selected gene are shown in the main view. Every gene has the **AA Chain** type, but some have no other region types at all; many have only a few region types. Text labels with more detailed information about protein regions appear on mouseover; this interaction is shown in the video included in the supplementary materials.

6.5 Variant and Variant Attribute Encoding

The goal for encoding variants and variant attributes was to show all of them at once; that is, to allow variant impact to be assessed with an information-dense overview that does not require interaction. Figure 6.3 summarizes the visual encoding choices.

Variants are encoded as vertical lines that traverse the entire transcript and protein regions. The lines have high visual salience, emphasizing the relationship of variants across the transcript and protein regions in one view.

The attributes for a variant are encoded at the top of its line, stacked vertically. Although the horizontal screen space is occupied by bars encoding the transcript and protein regions, there is considerable vertical screen space available. This scheme also allows attributes to be clearly associated with the variant without occluding the transcript and protein regions, leading to a primary display without visual clutter. The top of the stack has a two-part icon to show database status, with a small hollow circle on top if the variant appears in the *known-harmless* database and a small filled-circle icon just below it to show that is listed in a *known-cancer* database (Q4). A single variant could be in both of these databases simultaneously, so a different vertical region is allocated to each. Having a variant appear in both the *known-harmless* database and *known-harmful* database is possible because these databases are imperfectly curated. Just below, variant type is encoded with an icon (Q1); we use a set of 7 evocative icons culled from the biological literature. Resolving the variant type was difficult in previous tools because either it was not shown at all, or it was encoded with a very small mark such as a small, circular mark or thin line [4, 5, 9]. Moreover, the small size of these marks precludes the effective use of color coding [48] to show any other addition.

Below the variant type is the amino acid type for both the reference genome and the sample at that position (Q3). The 20 amino acid types are shown using single letters, following biological convention (Q2); we note that color coding is precluded since there are 20 choices. Changes in type are thus shown implicitly by having different symbols next to each other in

the stack.

A small grey arrow appears at the very top of the stack to distinguish the variants for a particular patient as needed for the clinical patient-focused task (Q10, Q11); the grey arrow marks are shown in Figure 7.3. We chose to use an additional mark to highlight rather than changing color to ensure that the color coding choices discussed below remain clearly visible.

We wanted variant hotspots to be highly salient (Q5). Our layout emphasizes recurrence of variants across samples by repeating the variant unit as many times as it recurs. The large region of encoded pixels created by this repetition results in a highly visually salient triangular visual footprint, as shown in Figure 6.4(d) and Figure 7.1(a); Figure 7.1(c) shows an example of a gene with no hotspots and no highly visually salient triangular visual footprint: the variants are more uniformly distributed. In contrast, previous work has shown recurrence in a way that is far more subtle, through position coding of a small object across a small position range, so it is easy to miss [4, 5, 9].

6.6 Use of Color

In the vertical stack of variants in the top part of the main view, we reserve the use of color for emphasizing changes of type implicitly. Amino acid chemical class is encoded with one of 4 different colors, red, green, light blue, and blue, so that a change of class is apparent as a change of color. These changes have a high impact, and so are encoded with high salience. The regions are relatively small, so we use high-saturation colors; we do take care to ensure that the text protein symbols in the foreground have sufficient luminance contrast to be visible. We chose colors to be highly distinguishable while still colorblind-safe through varying saturation and brightness.

In the bottom Transcript/Protein section, bars are colored if variants strike through them; otherwise they are shown in desaturated grey. The always-visible bars that stretch across most of the view each have their own color for memorability and visual salience: the Transcript bar is orange, and

the AA Chain protein region bar is green. All of the other bars are blue if they are struck by a variant (Q7).

We do reuse colors between the top and bottom parts of the view. While is it possible that the similarities between colors in the two parts of the view could be misinterpreted as implying a connection between data that is not in fact related, such as the **AA Chain** with the **Special AA** class, shown in the legends on the left side of Figure 6.3, we made a considered tradeoff. Our main goal was highly distinguishable colors in both places, with a secondary goal of a reasonably unified palette; the spatial separation between the parts of the view makes the misinterpretation less likely.

6.7 Design Comparison

We compare existing visual representations for variant analysis to our visual encoding to motivate the strengths of our design, showing the same variant data for a more direct and fair comparison. Figure 6.4 shows a comparison of variant data for the known gene DNMT3A between the encoding schemes of cBio [4], MuSiC [9], Ensembl variation image [5], and Variant View. Because of the usage barriers described in Chapter 5, the images from previous work are mockups created through close reading of the associated papers and personal communication with the authors. Our discussion focuses on the intellectual design considerations of each representation, not on the underlying implementation of the system or tool that generates them.

Both cBio and MuSiC encode variants as small colored circles on top of vertical lines that indicate their position on the protein coordinate, as shown in Figures 6.4(a) and (b). While MuSiC uses circle color to represent a limited number of variant types, neither representation shows the variant attributes of known database information or chemical class change, and only MuSiC shows AA change consistently. In both cases, the variant context of the transcript is absent and protein regions are represented as colored blocks all on the same vertical row, so there is a risk of occlusion. The high color saturation for these protein regions also tends to make them the centre of focus rather than the variants themselves. Figure 6.4(d) shows the Variant

View encoding. We argue that it is both more information-dense and more visually salient than these previous tools.

The Ensembl mockup in Figure 6.4(c) focuses on only the transcript and protein regions, with annotations to show where the many other alternative transcript and protein regions would take a great deal of additional vertical screen space. To avoid clutter, Variant View instead shows these alternative transcripts in separate views. In the Ensembl variation image, variants are encoded as thin lines, with variant type encoded as color which makes type difficult to resolve; in addition, occlusion of amino acid encodings can occur if the variants are close together. Variant lines can also overlap if they are in the same location, making it difficult to determine how many variants are present, and their variant type. Furthermore, the design relies on interaction in the form of scrolling and clicking to expose more information from the representation, instead of encoding it densely at the overview level. While we do not attempt to make an exact estimate of the speedup, we argue that analysts would need significantly less time to extract the important information from Variant View, where it is all shown immediately, then from the sprawling Ensembl variation image.

6.8 Implementation

Variant View was implemented using a combination of HTML, CSS, JavaScript, and the JavaScript Data-Driven Documents (D3) library [2]. We chose to deliver the tool as a web application to maximize accessibility and appeal for biologists, who find software installation a significant barrier to entry. The two versions of the prototype, Discover versus Compare, are accessible through different URLs. It is available as open source at <http://www.cs.ubc.ca/labs/imager/tr/2013/VariantView>.

In addition to the user-specified input file of variant data, Variant View accesses the UniProt database [44] for protein information and the RefSeq database [36] for exon information.

Figure 6.3: Variant visual encoding. Three variants that all occur at the same position are encoded in this diagram. Each variant has the identity of its type indicated by an icon, which is pointed to by the Variant Type label and arrow figure annotations. The reference amino acid is encoded by one of the twenty possible protein symbols. For instance, the reference amino acid pointed to by the Reference AA label and arrow figure annotations is an “S”, and represents the amino acid at this position in the reference genome. Below the reference amino acid is the variant amino acid. The variant amino acid is the identity of the amino acid that a particular sample, or individual, has. In this diagram, the symbol pointed to by the Variant AA label and arrow figure annotations is a “T”. Additionally, the amino acid chemical class is indicated by a colored circle. For instance, the reference “S” belongs to the “Special” amino acid class. A class color legend is shown in the figure. Additionally, whether the variant is reported in a known-harmless or known-harmful database is encoded by an empty circle, and/or a filled circle, respectively, at the top of the variant encoding. The transcript and protein regions the variants intersect are encoded as horizontal bars at the bottom of the display. Color and space is used to redundantly encode the difference between the important functional transcript and protein regions.

6.8. Implementation

(a)

(b)

(c)

(d)

Figure 6.4: Comparison of the same variant data between different visual encoding schemes. (a) cBio [4] mockup. (b) MuSiC [9] mockup. (c) Ensembl variation image [5] mockup. (d) Variant View screenshot.

Chapter 7

Case Studies

We now present three case studies which provide initial evidence that Variant View is useful to domain experts in several ways. First, it integrates diverse data types previously distributed across input files and external databases. Second, it provides summary metrics that are valuable for sorting genes and identifying candidates for further exploration. Third, it displays rich information about variant type and distribution across a gene. This information is not available in any other visualization tool and is valuable for interpreting the biological impact of variants, which requires human inspection.

The method of case studies is chosen as a validation technique over other methods such as head-to-head comparisons with previous work via benchmarks that show an improvement in task completion time. Case studies involve showcasing results that are found by target users through their use of the tool, and walking the reader through why the result images are effective through a qualitative discussion. The case study is a common method for presenting the results of a design study [28]. Munzner’s nested model [29] describes that the case study validation approach is strongest when there is an explicit discussion pointing out the desirable properties in the result images of the visualization tool’s use; in the discussion of results that follows we make sure to explicitly describe how Variant View exposes meaningful combinations of attributes to the target user to help them complete some of their tasks as defined by Table 4.2.

7.1 Case Study 1: Discover

Variant View consolidates transcript, protein and variant position and attributes into a single summary view, in contrast to the previous complex

workflow described in Section 4.2.1. The analysts used Variant View first for hypothesis confirmation, to see if the tool could expose known types and distributions of variants in genes implicated in AML, and then for hypothesis generation, to discover new variants that play a role in AML.

7.1.1 Hypothesis Confirmation

Upon sorting by the hotspot metric (Q9), the first three genes in the list were DNMT3A, IDH2, and FLT3. All of these have been reported in the literature as being affected by AML variants and this finding provides evidence that the tool can help confirm positive controls of the disease.

Once promising candidate genes were identified by simple sorting on summary metrics, our analysts then used the rich information available in the Variant View visualization to examine the variants' biological contexts. Figure 7.1(a) and (b) show the gene-level view of FLT3 and IDH2, respectively. The analysts found that the visual encoding in the main window was highly effective at emphasizing the hotspots at the gene level with visually salient bloom-like structures (Q5). They also noted how easily they could relate protein region information to variant position (Q7). In particular, Figure 7.1(a) reveals variant intersections with many different protein regions, which would be considerably more difficult to interpret in tabular format. In addition, Variant View exposes the diversity of variant types within a given hotspot. For example, the clusters in Figure 7.1(a) contain many different types of variants, whereas the cluster in Figure 7.1(b) is comparatively uniform in variant type. Our analysts were interested in such differences. These details are not captured by simple summary measures, like our hotspot metric, but rather require visual inspection and human interpretation. Overall, Variant View provided a notable acceleration of our analysts' previous manual workflow and they could see immediately what would have taken them at least 15 minutes to find.

7.1. Case Study 1: Discover

(a)

(b)

(c)

Figure 7.1: Confirming AML genes. Variant View allowed analysts to quickly confirm known results: known AML genes could be found near the top of the sorted lists, and the per-gene views clearly and immediately showed tell-tale structure. (a) IDH2. (b) FLT3. (c) Example gene without interesting structure near the list bottom.

7.1.2 Hypothesis Generation

In addition to retrieving and inspecting known variants in important AML genes, our analysts successfully used Variant View to discover interesting

7.2. Case Study 3: Compare

candidate genes. For example, Figure 1.2 shows one of these genes, and two more are shown Figure 7.2 (a) and (b). The gene names have been sanitized since their research is still ongoing and sample IDs in all examples have been sanitized to protect patient privacy. Figure 1.2 shows a concentration of variants that would be difficult to reveal in a spreadsheet or list interface. Just as with the hypothesis confirmation examples, Figure 1.2 and Figure 7.2 (a) and (b) reveal either uniform or diverse variant types within their hotspots in a way that is not communicated by the hotspot metric alone. Interpretation of the biological importance of this variant diversity requires human judgement, as does the significance of intersected protein regions. A1 remarked on the limitations of the previous workflow compared to using Variant View for Q5, Q6, and Q7:

It was really difficult to try and imagine the distribution of the variants along both the transcript and the protein - furthermore, the number of look ups required to determine whether the variants intersected important protein domains would have made searching all of them really difficult - getting extra detail about the protein regions would add an additional layer of workload.

7.2 Case Study 3: Compare

Analysts A3 and A4 used Variant View for the Compare Patient task, as described in Section 4.2.1. Figure 7.3 shows the immediate neighbors on each side of each variant, with the patient's own data indicated by the grey arrows at the top of the stack. It is immediately apparent that the leftmost and middle variants are exact matches with the known-AML variants on their left sides; reveal of the patient variants' neighbors is also demonstrated in the supplementary video. It is also immediately apparent that the rightmost variant does not have a match in the database: its neighbor is relatively far away and has very different attributes. The analysts remarked on how quickly the tool allowed them to draw these conclusions.

7.3. Case Study 4: Debug Pipeline

Figure 7.2: Discovering AML genes. Variant View has proved useful for analysts in the discovery process of identifying new candidate genes. (Gene names sanitized as their follow-up research is still ongoing.) (a) The clear hotspots indicated a gene of interest. (b) The fact that the variants strike a range in a known function region type was the most informative aspect of this layout.

7.3 Case Study 4: Debug Pipeline

The Debug Pipeline task, as discussed in Section 4.2.1, emerged later in our interactions with analysts and like the Compare Patient task it was suggested after presentations of the tool designed for the Discover Genes context.

Analyst A3 found spurious data from what he thought was a fully debugged pipeline when using Variant View. Figure 7.4 shows the surprising visual pattern for a gene (name sanitized). He quickly concluded that the sheer number of repeated identical variants that he saw was highly unlikely to reflect true dataset structure of the same variant occurring in so many different individuals. After solving this particular pipeline problem, A3 remarked:

7.3. Case Study 4: Debug Pipeline

Figure 7.3: Comparison of patient data to a known-AML variant database. The immediate neighbors for each variant are shown.

The tool exposed artifacts in the data that slid past at least two rounds of quality metric filtering, I was very surprised to see that there could be anything wrong with the data at this point - this type of problem would not have been caught by our previous, automated methods.

Figure 7.4: Debugging the bioinformatics pipeline.

Chapter 8

Discussion, Future Work, and Conclusions

In this section we discuss the design strategy of “specialize first, generalize later” as a way to tackle biological data visualization challenges. We also reflect more generally on the visualization design issues. We discuss our design progression through a description of the 8 prototypes we constructed. We discuss our design process with reference to some of the 32 pitfalls proposed by the authors of the design study methodology by Sedlmair et al. [38]. We then discuss limitations of the work with an emphasis on the design’s scalability, and summarize the conclusions of this work.

8.1 Specialize First, Generalize Later

The domain of biology has been a frequent target of design studies in visualization [7, 17, 24, 25, 27, 32]. We conjecture that this domain is a rich source of problems exactly because of its difficulty: there is an enormous amount of data to contend with, and figuring out what matters is nontrivial. In the language of the four-level nested model of visualization design [29], developing the appropriate *data abstraction* is a major part of the problem.

By abandoning whole genome coordinates and committing to transcript and protein coordinates, we created a specialized tool that targets key tasks in variant analysis, but does not offer the generality of a genome browser. We made this decision knowingly, and throughout the design phase we purposefully strove to optimize the display to the target tasks and did not require ourselves to produce a very general solution. This philosophy to specialize first has emerged from examination of many design studies [32, 38]; it seems

to be well suited for domains where the amount of up-front detail is enormous and it can be difficult to judge which design elements will generalize.

Generalization follows naturally from this initial specialization. We have found that opportunities for generality naturally emerge when analysts try out working prototypes on their own data, even though they are not obvious at the outset. For example, our original design targeted the Discover Genes task, but it later became apparent that Variant View could support the other two tasks with only minor adjustments. Additional applications and adaptations continue to emerge as we expose more analysts to the tool. For example, another group is interested to use Variant View to visualize variants in non-exonic regions, which are excluded in the current tool by our choice of coordinate system. A more general alternative to committing to transcript and protein coordinates would be to enable an analyst to define coordinates of interest: for example, non-exonic regions. Overall, this approach ensures that the decisions of what to generalize are guided by real-world use cases.

8.2 Visualization Design Considerations

We now reflect on our design choices by framing them in an abstract way that is not tied to the vocabulary of the domain problem. These choices can be organized into: *What to show* and *How to show it*.

What to show: A major abstract choice in this study was to *identify scales of interest within the data*. As discussed previously, the final choice of scales may break with convention, but should best serve the analysis task. Closely coupled choices were to *identify what data can be filtered out as being irrelevant* and to *determine what additional data to derive*. There is a tendency to display all information within the provided input file, but more often than not, much of that material is not useful to the target tasks and valuable derived metrics are missing.

How to show it: At several points in the design phase, we explored options for how to highlight a change in data value and the choice required: *deciding when comparison can be accommodated implicitly by visually encoding values through side-by-side marks versus by explicitly computing a*

value difference that is visually encoded directly. Although the side-by-side approach may introduce more visual clutter than a single difference value, it preserves the underlying data and may be the better choice for some tasks. A related choice concerned *deciding what to visually encode directly versus what to support through interaction.* Attempting to encode all pertinent data attributes can lead to visual clutter, but requiring extensive interaction can be taxing to the user. Similarly, navigation within a view can be very time consuming and we carefully considered when to *reduce navigation drastically or eliminate it completely.* Taken together, our approach regarding how to show the data was to create a multi-scale non-contiguous overview that showed all information without the need to navigate.

8.3 Design Progression

The design and implementation stages for this design study were tightly interwoven, with a series of 8 prototypes of increasing complexity created over five months. We decided that data sketches [20] were more appropriate than paper prototyping due to the complexity of the data, so even the earliest prototypes did load and show real data. The first two prototypes were static tests of visual encoding possibilities, where we received feedback by demonstrating them to the analysts; these early prototypes are shown in Figure 8.1(a) and (b).

The deploy stage began in the third month with the third prototype, which supported interactive search, shown in Figure 8.2, and a fourth prototype with different interactive filtering capabilities, shown in Figure 8.3.

Figure 8.4 shows a final design for a series of prototypes that allowed for interactive filtering. The three histograms at the top of the screen can be brushed with the mouse cursor to specify a range of quality values. Each histogram is a scented widget [49]; in addition to allowing the user to specify ranges, the scented widget shows the distribution of the data in order to provide information scent. Below these scented widgets are check boxes that filter the data on variant type. We abandoned this design because our collaborators had already pre filtered their data, so they did not need to

8.3. Design Progression

Figure 8.1: The first two data sketch prototypes. (a) shows an early data sketch that shows full genomic coordinates for the gene representation. The exon-containing gene representation is next to the “Gene:” title; exon regions are represented as skinny, vertical, dark grey bars, and are comparably small to the long, thin horizontal line of the inter-exon regions. Red vertical lines represent variants. (b) shows a representation that removes inter-exon regions to emphasize the formerly comparably small exons regions in (a). The transcript is colored orange, and protein regions are represented by bars in separate horizontal tracks below the transcript, each distinguished by color and spatial position. Variants are encoded by blue vertical strikes with blue circles on top.

specify these ranges.

Figure 8.2: An interactive search supporting prototype. This prototype retained the transcript and protein representations of Figure 8.1(b), but also allows the user to search for genes of interest interactively. The prototype also has the capability of interactive filtering on various variant quality and type attributes.

8.3. Design Progression

Figure 8.3: A second interactive search supporting prototype. This prototype retained the transcript and protein representations of the prototype shown in Figure 8.1(b), but also allows the user to search for genes of interest interactively. The prototype also has the capability of interactive filtering on various variant quality and type attributes; the prototype differs from the prototype in Figure 8.2 slightly in terms of layout, and mostly in terms of the attributes it allows the user to interactively filter data by.

Figure 8.4: An intermediate prototype for the Variant View design. The histogram interfaces at the top are brushable controllers that help define a threshold for filtering on a quality metric. The checkboxes allow the analyst to filter variants based on the variant types described in Section 4.1.3. Variants appear as vertical, blue lines that intersect an orange transcript stacked above multiple, colored protein regions below. Variant recurrence at a position is encoded by a large histogram stacked above, and aligned with, the transcript and protein regions.

8.3. Design Progression

One of the major problems with the first five prototypes is that although we show the necessary levels of biological context described in Section 4.1.2 that our analysts are interested in, we do not show the important variant attributes described in Section 4.1.3. Variants are encoded simply with vertical lines, and the only aspect analysts could resolve about each line is the variant's position along the transcript and protein regions. From our data and task abstraction described in Section 4, we decided to expose all variant attributes necessary for variant impact analysis within the gene view. The first prototype to show variant type information, one of the most important attributes for variant impact assessment, is shown in Figure 8.5. In this prototype, variant type is encoded with icons. We decided to remove the orange transcript since most of our discussion in interviews centred around whether variants intersected protein regions.

Figure 8.5: A sixth data sketch prototype with icons for variant type. This prototype retained the protein representations of the initial prototypes, but not the transcript representation. Additionally, the prototype includes icons representing the attribute of variant type for each variant.

8.3. Design Progression

Our analysts found the icon representation for variant type very useful. So, we used our data abstraction to decide which attributes to encode for each variant in the next prototype. Amino acid identity and class identity at each variant position were the next most important attributes to encode. One aspect of previous prototypes our analysts did miss, however, was the orange transcript representation, so we made sure to include it in the next prototype. In the seventh prototype, shown in Figure 8.6, amino acid change is shown using one of twenty protein symbols, and amino acid class is encoded with one of four colors in a circle surrounding the protein symbol.

Figure 8.6: A prototype showing protein symbols and amino acid classes. This prototype additionally encodes amino acid change and amino acid class type as a letter symbol and circle color, respectively. The orange transcript representation is once again included.

Our analysts were happy with the representation at this stage in the

design, but wanted more assistance navigating through the long list of genes that variants may occur in. Up to this point, the prototypes required that analysts know which gene they wanted to inspect *a priori*; they then had to type the gene name into the search box to display it. Based on interviews, we found that we could use derived measures of variant impact to sort the gene list ranked by these metrics. These metrics are described in Section 4.1.4. We created a list interface for the gene names, and allowed the user to sort by these gene names in the next prototype, whose display is the final prototype, shown in Figure 6.1. Finally, we made changes to the color palette of this prototype to ensure the amino acid class colors were distinguishable for people who are red-green color-blind.

8.4 Design Study Methodology Pitfalls Analysis

We discuss the current study in the context of selected pitfalls (PF) observed in the work of Sedlmair et al. [38]: the Design Study Methodology (DSM), shown in Figure 3.1. Figure 8.7 is an itemized list of the 32 pitfalls and is taken from [38]. We discuss the stages of the DSM where we feel our design study encountered turbulence due to pitfalls, or why and how our design process succeeded in avoiding certain pitfalls.

8.4.1 Pitfalls in the Winnow Stage

In the winnow stage, the goal is to identify and select the most promising collaborators to design a visualization solution for. At this stage, premature commitment to collaborators who either have no time to discuss their problem, or cannot give access to data, can sink a promising design study. In our study, we turned down three potential collaborations due to their unavailability before committing to a final group of analysts. We feel we avoided PF-3, “premature commitment; collaboration with the wrong people,” largely due to knowledge of PF-4, “no real data available (yet),” and PF-5, “insufficient time available from potential collaborators.” These pitfalls, along with the rest of the 32 pitfalls, are shown in Figure 8.7.

8.4. Design Study Methodology Pitfalls Analysis

PF-1	premature advance: jumping forward over stages	general
PF-2	premature start: insufficient knowledge of vis literature	learn
PF-3	premature commitment: collaboration with wrong people	winnow
PF-4	no real data available (yet)	winnow
PF-5	insufficient time available from potential collaborators	winnow
PF-6	no need for visualization: problem can be automated	winnow
PF-7	researcher expertise does not match domain problem	winnow
PF-8	no need for research: engineering vs. research project	winnow
PF-9	no need for change: existing tools are good enough	winnow
PF-10	no real/important/recurring task	winnow
PF-11	no rapport with collaborators	winnow
PF-12	not identifying front line analyst and gatekeeper before start	cast
PF-13	assuming every project will have the same role distribution	cast
PF-14	mistaking fellow tool builders for real end users	cast
PF-15	ignoring practices that currently work well	discover
PF-16	expecting <i>just talking</i> or <i>fly on wall</i> to work	discover
PF-17	experts focusing on visualization design vs. domain problem	discover
PF-18	learning their problems/language: too little / too much	discover
PF-19	abstraction: too little	design
PF-20	premature design commitment: consideration space too small	design
PF-21	mistaking technique-driven for problem-driven work	design
PF-22	nonrapid prototyping	implement
PF-23	usability: too little / too much	implement
PF-24	premature end: insufficient deploy time built into schedule	deploy
PF-25	usage study not case study: non-real task/data/user	deploy
PF-26	<i>liking</i> necessary but not sufficient for validation	deploy
PF-27	failing to improve guidelines: confirm, refine, reject, propose	reflect
PF-28	insufficient writing time built into schedule	write
PF-29	no technique contribution $\neq$ good design study	write
PF-30	too much domain background in paper	write
PF-31	story told chronologically vs. focus on final results	write
PF-32	premature end: win race vs. practice music for debut	write

Figure 8.7: Design Study Methodology: 32 pitfalls. Figure taken from [38] courtesy of Tamara Munzner.

Some problems, after thorough characterization in terms of the data and tasks required to solve them, may be completely automated and not require a visualization solution at all. We encountered this problem with one of our

first potential collaborators. Their problem involved detecting topological features in graphs. The problem seemed promising for a visualization approach; however, as the problem characterization, data, and task abstraction stages progressed, it became clear that a number of off-the-shelf methods including a simple prioritized list interface could solve their problem. Mistaking a problem that can be solved entirely by automated methods for an interesting visualization problem is characterized by the DSM's PF-6: "no need for visualization: problem can be automated."

We feel we mostly avoided PF-7: "researcher expertise does not match the domain problem": two of the three visualization researchers had ties to the domain of biology in the form of university training, so fundamentals for understanding the problem domain were present, and the interest level was high. When it came to the more specific domain of sequence variant analysis, however, this pitfall was not completely avoided due to some specifics of the domain: for instance, questions of, "what artifacts are present in sequence data?", and, "what attributes are of interest to sequence variant analysts?" This gap in knowledge was largely compensated for by the availability of our analysts and their willingness to meet to clarify or explain their workflow and domain. This availability was probably a result of having collaborators with a high level of interest in the visualization solution. In this collaboration, we feel we did not succumb to PF-11, "no rapport with collaborators," and a good rapport with our collaborators probably facilitated their continued support and feedback.

PF-8: "no need for research: engineering vs. research project," was a very apparent pitfall at multiple stages of this design study; carefully scoping the project to a specific visualization design problem helped avoid this pitfall. PF-8 captures situations wherein a problem can be solved with a system composed of known, not necessarily novel approaches; visualization research contributions from this process may either be minor or nonexistent. The computationally intensive filtering stages required to narrow down a list of sequence variants to a manageable size for analysis, mentioned in Chapter 2, is a difficult engineering problem; its solution would probably not contribute much to the field of visualization. Observing PF-9, "no need

for change: existing tools are good enough,” encouraged a survey of the field that exposed existing tools such as MedSavant [45] that could satisfy this filtering problem for our collaborators .

A final pitfall in the winnowing stage is PF-10: “no real/important/recurring task.” This pitfall aims to steer visualization design study practitioners away from investing time and effort in design solutions for minor and/or infrequent problems in an analyst’s workflow. An example of this pitfall occurred during our winnowing process: we were considering a collaborator who had immediate access to a wealth of promising microRNA data. We were enthusiastic at the prospect of designing a visualization tool for this data, but after additional talks with the potential collaborator, we found that there were no front-line analysts available to work with the data; furthermore, the potential collaborator was busy just trying to generate more raw data, and did not have time to interpret and analyze it at length. Because we had data but no tasks to design a solution for, we parted ways.

The central goal of the winnow stage is to identify the most promising collaborations [38]. In our study, we found that the winnow stage initiated evaluation of whether or not a certain collaboration would result in an effective visualization tool. The concept of a winnow stage also helped us to start thinking about project scope and planning: identifying collaborators able to articulate data and task descriptions made it easier to design a visualization tool within a reasonable scope of time. Definition of pitfalls in the DSM helped us to identify more concrete scenarios that should be avoided.

However, some pitfalls defined within the winnow stage of the DSM were more difficult to identify than others. We found PF-4, PF-5, PF-7, PF-11, shown in Figure 8.7, the easiest to identify early in the winnow stage without progressing to more advanced stages of the design process. We found PF-6, PF-8, PF-9, and PF-10 much more difficult to reliably identify without a deeper knowledge of the analysts’ data and tasks. In the DSM, data and task abstraction does not occur until the design phase. Based on our experience with this methodology, we feel that spending time abstracting data and tasks can help determine whether a problem can be automated (PF-6), whether a problem requires an engineering effort but no interesting

research contributions are possible (PF-8), whether existing tools are good enough for the current problem (PF-9), and whether the task is important enough or occurs frequently enough that investing in a visualization solution is worth the time (PF-10).

8.4.2 Pitfalls in the Cast Stage

Characterizing the roles collaborators play during the design study process is part of the cast stage. This stage is important because it can help determine who the visualization solution is being designed for and other considerations such as permission to use potentially sensitive or private data.

The first of the pitfalls associated with this stage is PF-12: “not identifying front-line analyst and gate keeper from start.” A visualization solution should be developed specifically for the front-line analysts’ data and tasks. Knowledge of this pitfall helped us to identify the front-line analysts whose problem included performing tasks on the data described in Section 4.1. This pitfall may seem like an obvious one to avoid, but often there are other collaborators who can be mistaken for front-line analysts based on their vast knowledge of the problem domain, or knowledge of or familiarity with the tasks of the actual front-line analysts. In our design study, a fellow computational tool builder was an example of this cast member: they had an expert level of knowledge regarding the variant data generation phase, which was one of their principal responsibilities, but through exposure to, and interaction with front-line analysts, they came to understand some of the data and tasks they were performing. In the end, they themselves would not be using the tool for variant analysis, so we ensured we tailored the tool for the collaborators performing direct analysis on the variant data first. This decision helped us avoid PF-14: “mistaking fellow tool builders for real end users.” It was also important to recognize that one collaborator could take on many roles. We found that the fellow tool builder mentioned was also a potent *connector*: they were able to connect us with the front-line analysts, the principal investigator for their lab, and another fellow tool builder that generated variant data. The principal investigator of the lab was able to

give us access and permission to use certain data, and have time with the analysts. In the language of the DSM, the principal investigator was a *gate keeper*, and becoming acquainted with them early helped us avoid PF-12: “not identifying front-line analyst and gate keeper from start.” Neglecting to meet with and/or get permission from the gate keeper during the course of the design study can lead to problems with access to data, problems with access to analysts, and problems related to the responsible release of information in publications.

One observation we made during the course of this design study that supplements the description of cast and the operational consequences of casting certain collaborators in the DSM came after we were happy with a near-final design for the system: after checking in with the front-line analysts, we went back to our fellow tool builder collaborators to get their input. What we found is that they could use the tool for spotting artifacts in the variant data that could be a result of the variant data generation pipeline described in Chapter 2. This idea is somewhat captured by PF-13: “assuming every project will have the same role distribution,” because the fellow tool builder had in some sense become a front-line analyst for a different task: the Data Debug task.

8.4.3 Pitfalls in the Discover Stage

The discover stage of the DSM is related to requirements analysis in software engineering. To adequately characterize the problem front-line analysts are dealing with, it is necessary to extract information from them. This process is usually iterative.

During this information extraction stage, one pitfall is captured by PF-16: “just talking and fly-on-the-wall.” Just talking to users is necessary but often not sufficient to extract information about their data and tasks: often what a target user says they do in retrospect is only an incomplete match with their actual activities. Another common observation technique is fly-on-the-wall, wherein the researcher silently and unobtrusively observes the target user complete their tasks in their habitual environment. In our study,

we began with a semi-structured interview, and then constructed prototypes between each interview to present at the subsequent one to garner feedback; this process is described in Chapter 3. We found that constructing prototypes that could load real data helped us to avoid mistaking the structure and scale of the true datasets. This process of creating data sketches [20] versus paper prototypes is valuable even at the very early stages of the design process. This process also invited more feedback from the collaborators, who could test out the interface and comment on what data attributes they wished they could see, but could not, and other feedback. The demo sessions helped guide the design away from solutions we had first thought were quite final: for instance, one design, shown in Figure 8.4, depicts the variants as blue lines across a transcript and protein annotations. During a demo session of this tool, analyst A1 mentioned that: “This representation looks good, but I know there’s supposed to be a stop mutation in this gene. Which one is it? Why can’t I see it?” A stop mutation is one of the variant types described in the Data Abstraction of Section 4.1. Exposure to these prototypes helped refine and prioritize the data our analysts were really interested in. Furthermore, the demo sessions could fit into less than one hour time slots during the week, which could be more convenient and time efficient than other invasive methods of watching and interrupting the analysts to ask questions as they worked. Analyst time can be precious: in our study, analysts met with us in addition to their myriad day-to-day responsibilities. It was important to keep interaction with them brief and meaningful. Moreover, seeing iterative prototypes appeared to engage their interest as they could see their feedback manifest itself either in the choice of visual encodings or interaction techniques available in each successive design.

Another pitfall we encountered during the problem characterization stage was PF-17: “experts focusing on visualization design vs. domain problem.” This pitfall captures instances wherein target analysts focus on communicating their problems in the form of either new or previous design solutions instead of communicating their domain problems directly. For instance, a potential collaborator explained that their domain problem could be solved

with a particular graph visual encoding of their data, without specifying the problem they were trying to solve with their data. Echoing previous solutions to this pitfall, we also found that focussing on the analysis problem directly, and providing the target analyst with prototype design alternatives, helps tease them apart from design solutions they are fond of or assume will be effective.

A final pitfall encountered during the discover stage is PF-18: “learning their problems/language: too little/too much.” This pitfall attempts to guide design study practitioners towards a balance of enough domain problem/language knowledge required to understand the data and tasks involved, without becoming so immersed in the problem domain that domain-specific details creep into the data and task abstraction stages. Having some biological knowledge *a priori*, we were confident about our understanding of the biological transcript and the attributes and annotations analysts might like to see. However, it was not obvious to us that our final collaborators did not want to see variants that occurred outside of exon regions. This instance was an example of having too little knowledge of the problem domain. Later in the study, during initial writing stages, the data and task abstraction included too much domain information: for instance, we included just, “variant mutation types,” and neglected to specify that these were, “categorical attributes,” and that there were a total of about seven of them. This underdeveloped data abstraction made it difficult to motivate why we used the visual encoding of an icon. There are preferable ways of encoding categorical attributes: for instance, hue or icon. Furthermore, knowing the number of categories or the range of values an attribute can exhibit can also motivate their encoding. For instance, if an attribute can only exhibit one of a small, finite number of categories, encoding category with hue is a sound design decision because less than one dozen colors are distinguishable when showing categorical data [48].

8.4.4 Pitfalls in the Design Stage

Once the domain problem has been thoroughly characterized, the design study researcher can begin designing a visualization solution. According to the DSM, the design stage includes generation of data abstractions, visual encodings, and interaction mechanisms [38]. A first pitfall in this stage is PF-19: “abstraction: too little.” In the previous section we fell into a pitfall that lead to not enough abstraction: this problem came from PF-18, and letting too much domain specific language creep into the data abstraction. Before this specification was resolved, it limited our ability to think clearly about the data we needed to present to the analysts, and successfully connect the theory of visual encoding selection to attribute type: for instance, some visual encodings are more effective for sequential data (can encode with intensity) than categorical (can encode with hue).

Since our design study process incorporated a wealth of prototypes and sketches for the design of Variant View, we feel that we mostly avoided the DSM’s PF-20: “premature design commitment: consideration space too small.” However, the tool used for the Confirm and Discover tasks are slightly different than the tool used for the Compare task. The Compare task tool presents known variant data to the left and right of a patient variant dataset, and is discussed in Section 7.2; the tool is shown in Figure 7.3. The design cycle for the Compare task tool was much shorter, and we feel that given more time to present prototype design alternatives we could help this design mature and avoid PF-20.

8.5 Scalability Limitations

Variant View supports the display of up to 52 variants per gene on a 1280 by 800 pixels display (primary view: 675 pixels wide; each variant encoding: 13 pixels wide). Above 52 variants, the display scrolls horizontally to show additional variants. This scale choice is appropriate for our target analysts’ datasets, which undergo a previous filtering step in their workflow. We initially experimented with supporting the filtering stage within Variant View

itself. Existing tools, such as cBio [4] and MuSiC [9] have similar display limits, which we estimate at 80 and 60 variants, respectively, onscreen without overlaps. The Ensembl variation image [5] and other genome browsers can display hundreds of variants because they are encoded as thin vertical lines, but overlaps and occlusion can lead to difficulties in determining whether there is one or many variants at a single position.

8.6 Future Work

A major limitation of the current tool is the number of variants that can be displayed in the primary view at once. If the number of variants per gene did increase above 52, or much larger, there are two families of methods for reducing the amount of information shown: *Item reduction methods* and *attribute reduction methods*.

Item reduction methods are pervasive in the HCI and visualization literature; their goal is to reduce the number of items that need to be shown onscreen. Two major methods of item reduction are *filtering* and *aggregation*. Navigation, such as panning and or zooming in to see fewer items are suggested to be a special case of filtering. The limitations and time costs associated with zooming are discussed in Section 6.2. Filtering and aggregation are therefore fruitful avenues of exploration.

One possible area of future work would be to directly add visualization support for the variant data filtering stage mentioned in Chapter 2. Addition of filtering was attempted at one stage of the design process, and is shown in Figure 8.4. The three histograms at the top of the screen can be brushed with the mouse cursor to specify a range of quality values. The histogram is a scented widget [49]; in addition to allowing the user to specify ranges, the scented widget shows the distribution of the data in order to provide information scent. Below these scented widgets are check boxes that filter the data on variant type. We abandoned this design because our collaborators had already pre filtered their data, so they did not need to specify these ranges. The approach also fell short because the task requires a considerable engineering effort to support manipulation of very large (10,000 item) data

sets in a time frame that would be acceptable to the user. After attempting this approach we felt that it fell outside the scope of designing a solution for a visualization research problem. Furthermore, we found that tools such as MedSavant [45] already target interactive filtering and manipulation of large variant data sets, but lack the capability to depict the variant attributes designed in Section 4.1 within the context of a gene transcript and protein annotations. One future direction could be to integrate Variant View with MedSavant and allow its back end to support filtering large variant datasets based on various quality metrics. We have considered MedSavant over other tools because we have conducted initial contact with its developers who have expressed interest in a collaboration. This collaboration did not take place during the course of this design study due to the constraints of time and the logistics of coordinating with off-site collaborators.

In contrast to the filtering method of item reduction, the aggregation method involves summarizing information about data items with marks that represent several underlying items instead of only one item. Aggregation can be used to construct overviews of data. Since our analysts would prefer to see as much variant information as possible simultaneously, aggregation might be a more appropriate method than filtering, because we can use our data and task abstractions to prioritize a list of attributes for creating marks that adequately represent several underlying variant items instead of just a single one. The challenge of aggregation is avoiding elimination of interesting signals in the dataset when trying to summarize the data. Determining a priority for attributes could help alleviate this problem. In our design study, variant type was the most important attribute for determining variant impact. After variant type, analysts considered other factors such as variant recurrence. Creating a successful mark in this case should therefore incorporate variant type. Since we deemed that variant recurrence was the second most important attribute, it should also be reflected in this mark. Given the current design of Variant View, the scarcest resource for displaying information is horizontal screen space. If there are many variants, we will run out of room in the horizontal, or x-axis, screen dimension. A particular instantiation of a tool that encodes information

horizontally, but applies hierarchical aggregation to summarize information in the vertical direction is by Stolte et al. [40].

The tool is shown in Figure 8.8, and encodes time series data for debugging superscalar processor performance problems. A horizontal timeline view is encoded as a four-tiered strip chart; there are four levels of aggregation. Starting from the bottom strip up, an analyst can view data at increasing levels of detail. This aggregation approach might be suitable for variant visualization since the bottom strip can preserve spatial information of where along a gene the variant occurs, while the upper strips could encode more attributes of the variant at regions specified on the bottom strip.

Figure 8.8: A hierarchical Aggregation Example. Multiple levels of aggregation are used to provide multiple levels of overview in this timeline view for debugging superscalar processor performance. From [40], Figure 1. Figure courtesy of Christopher Stolte

Another possible avenue for future work is the refinement of the Variant View Compare task tool which is shown in Figure 7.3. This design was much more preliminary than the design created for the Confirm and Discover tasks, because they were part of the original problem characterization phase. There are a few apparent considerations for this tool such as whether or not it is correct to show just the variants to the right and to the left of the patient's variant data in this tool's neighborhood view. It could be that a

patient’s variant is dismissed as being not harmful because we only show just the variants to the left and right it. This problem could be solved through further discussions with the analysts, who might give us a more realistic threshold number for neighbourhood size of variants around the patient variant. It could also be solved by having a tuneable control in the form of a slider that interactively allows for neighbourhood threshold resizing. This project is an interesting avenue, but future work in this area will take more gatekeeper approval considering the sensitivity of the patient data in this domain.

8.7 Conclusions

In this design study we designed, implemented and deployed a tool for genetic variant impact assessment. One contribution of this thesis is a data and task abstraction for the problem domain of variant analysis: our task analysis links concrete, domain-specific questions to this data abstraction, as described in Chapter 4. Another contribution is a discussion in Section 6.2 that reflects on the strengths and weaknesses of genomic coordinates as a data abstraction, a question that has broad implications for the design of biological visualizations. A third contribution is the validated design and implementation of Variant View described in Chapter 6 and Chapter 7. We carefully justify our choices for visual encoding and interaction techniques with respect to the data and task abstractions. We validate the effectiveness of the tool with three case studies of its use after several months of deployment in Chapter 7. Our final contribution is a discussion in Chapter 8 of the lessons learned in this design study: the design strategy of “specialize first, generalize later”, and six design considerations organized into the themes of “what to show” and “how to show it”. Variant View was originally designed for the specific variant analysis task of Discover Genes in collaboration with two analysts, but we were able to adapt the design with minimal changes to two additional tasks for other analysts. The combination of thorough data abstraction and task analysis led us to select and prioritize data in this domain in terms of what should be emphasized, de-emphasized, or completely

discarded. Our goal was an information-dense overview showing multiple, non-contiguous features at multiple scales. We succeeded in designing a main view that did not require any navigation, and limited our use of interactivity to simple techniques of sorting secondary views and bidirectional linking between views. In contrast, previous tools in this domain rely on interaction techniques that are costly in terms of both speed of execution and mental workload, or present an incomplete view of the dataset, so that some user questions could not be answered.

Bibliography

- [1] A. Altmann, P. Weber, D. Bader, M. Preuss, E. B. Binder, and B. Mller-Myhsok. A beginner’s guide to SNP calling from high-throughput DNA-sequencing data. *Human Genetics*, 131(10):1541–1554, 2012.
- [2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis 2011)*, 2011.
- [3] T. Carver, S. R. Harris, M. Berriman, J. Parkhill, and J. A. McQuillan. Artemis: an integrated platform for visualization and analysis of high-throughput sequence-based experimental data. *Bioinformatics*, 28(4):464–469, 2012.
- [4] E. Cerami, J. Gao, U. Dogrusoz, B. E. Gross, S. O. Sumer, B. A. Aksoy, A. Jacobsen, C. J. Byrne, M. L. Heuer, E. Larsson, Y. Antipin, B. Reva, A. P. Goldberg, C. Sander, and N. Schultz. The cBio cancer genomics portal: An open platform for exploring multidimensional cancer genomics data. *Cancer Discovery*, 2(5):401–404, 2012.
- [5] Y. Chen, F. Cunningham, D. Rios, W. M. McLaren, J. Smith, B. Pritchard, G. M. Spudich, S. Brent, E. Kulesha, P. Marin-Garcia, D. Smedley, E. Birney, and P. Flicek. Ensembl variation resources. *BMC Genomics*, 11(11), 2010.
- [6] P. Cingolani, A. Platts, M. Coon, T. Nguyen, L. Wang, S.J. Land, X. Lu, and D.M. Ruden. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of *Drosophila melanogaster* strain w1118; iso-2; iso-3. *Fly*, 6(2):80–92, 2012.

- [7] P. Craig and J. Kennedy. Coordinated graph and scatter-plot views for the visual exploration of microarray time-series data. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 173–180, 2003.
- [8] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, and R. Durbin. The variant call format and VCFtools. *Bioinformatics*, 27:2156–2158, 2011.
- [9] N.D. Dees, Q. Zhang, C. Kandoth, M.C. Wendl, W. Schierding, D.C. Koboldt, T.B. Mooney, M.B. Callaway, D. Dooling, E.R. Mardis, R.K. Wilson, and L. Ding. MuSiC: Identifying mutational significance in cancer genomes. *Genome Res.*, 22:1589–1598, 2012.
- [10] M. N. Edmonson, J. Zhang, C. Yan, R. P. Finney, D. M. Meerzaman, and K. H. Buetow. Bambino: a variant detector and alignment viewer for next-generation sequencing data in the SAM/BAM format. *Bioinformatics*, 27(6):865–866, 2011.
- [11] M. Fiume, V. Williams, A. Brook, and M. Brudno. Savant: genome browser for high-throughput sequencing data. *Bioinformatics*, 26(16):1938–44, 2010.
- [12] S. A. Forbes, N. Bindal, S. Bamford, C. Cole, C. Y. Kok, D. Beare, M. Jia, R. Shepherd, K. Leung, A. Menzies, J. W. Teague, P. J. Campbell, M. R. Stratton, and P. A. Futreal. COSMIC: mining complete cancer genomes in the Catalogue of Somatic Mutations in Cancer. *Nucleic Acids Res.*, 39(Database issue):945–950, Jan 2011.
- [13] G. W. Furnas. Effective view navigation. In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pages 367–374, 1997.
- [14] H. Hou, F. Zhao, L. Zhou, E. Zhu, H. Teng, X. Li, Q. Bao, J. Wu, and Z. Sun. MagicViewer: integrated solution for next-generation sequencing data visualization and genetic variation detection and annotation. *Nucleic Acids Res.*, 38(Web Server issue):732–736, 2012.

- [15] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.
- [16] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The human genome browser at UCSC. *Genome Res.*, 12(6), 2002.
- [17] R. Kincaid, A. Ben-Dor, and Z. Yakhini. Exploratory visualization of array-based comparative genomic hybridization. *Information Visualization*, 4(3):176–190, 2005.
- [18] H. Lam. A framework of interaction costs in information visualization. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis 2008)*, 15(6):1149–1156, 2008.
- [19] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25:2078–2079, 2009.
- [20] D. Lloyd and J. Dykes. Human-centered approaches in geovisualization design: investigating multiple methods through a long-term case study. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2011)*, 17(12):498–507, 2011.
- [21] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytzky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, 20:1297–1303, 2010.
- [22] P. McLachlan, T. Munzner, E. Koutsofios, and S. North. LiveRAC: interactive visual exploration of system management time-series data. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pages 1483–1492, 2008.

- [23] M. L. Metzker. Emerging technologies in DNA sequencing. *Genome Res.*, 14:1767–1776, 2005.
- [24] M. Meyer, T. Munzner, A. dePace, and H. Pfister. MulteeSum: A tool for exploring space-time expression data. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2010)*, 16(6):908–917, 2010.
- [25] M. Meyer, T. Munzner, and H. Pfister. MizBee: A multiscale synteny browser. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2009)*, 15(6), 2009.
- [26] M. Meyer, M. Sedlmair, and T. Munzner. The four-level nested model revisited: blocks and guidelines. In *Proc. VisWeek Workshop on Beyond time and errors: novel evaluation methods for Information Visualization (BELIV)*, 2012.
- [27] M. Meyer, B. Wong, M. Styczynski, T. Munzner, and H. Pfister. Pathline: A tool for comparative functional genomics. *Computer Graphics Forum (Proc. EuroVis 2010)*, 29(3):1043–1052, 2010.
- [28] T. Munzner. Process and pitfalls in writing information visualization research papers. *Information Visualization: Lecture Notes in Computer Science*, 4950:134–153, 2008.
- [29] T. Munzner. A nested model for visualization design and validation. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis 2009)*, 15(6):921–928, 2009.
- [30] C. Nielsen and B. Wong. Points of view: Managing deep data in genome browsers. *Nature Methods*, 9(6):521–521, 2012.
- [31] C. Nielsen and B. Wong. Points of view: Representing the genome. *Nature Methods*, 9(5):423–423, May 2012.
- [32] C. B. Nielsen, S. D. Jackman, I. Birol, and S. J. M. Jones. ABySS-Explorer: Visualizing genome sequence assemblies. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2009)*, 15(6):881–888, 2009.

- [33] S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, M. R. Speicher, J. Zschocke, and Z. Trajanoski. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefings in Bioinformatics*, 13(10):1–26, 2013.
- [34] P. Pirolli and S. K. Card. Information foraging. *Psychological Review*, 106(4):643–675, 1999.
- [35] M. Plumlee and C. Ware. Zooming versus multiple window interfaces: Cognitive costs of visual comparisons. *Proc. ACM Trans. on Computer-Human Interaction (ToCHI)*, 13(2):179–209, 2006.
- [36] K. D. Pruitt, T. Tatusova, and D. R. Maglott. NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Res.*, 35(Database issue):61–65, Jan 2007.
- [37] F. Sanger and A.R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology*, 94(3):441–446, 1975.
- [38] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis 2012)*, 18(12):2431–2440, 2012.
- [39] S. T. Sherry, M. H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin. dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res.*, 29(1):308–311, Jan 2001.
- [40] C. Stolte, R. Bosch, P. Hanrahan, and M. Rosenblum. Visualizing application behavior on superscalar processors. In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pages 10–17, 1999.
- [41] J. A. Tennessen, A. W. Bigham, T. D. O’Connor, W. Fu, E. E. Kenny, S. Gravel, S. McGee, R. Do, X. Liu, G. Jun, H. Min Kang, D. Jordan, S. M. Leal, S. Gabriel, S. J. Rieder, G. Abecasis, D. Altshuler,

- D. A. Nickerson, E. Boerwinkle, S. Sunyaev, C. D. Bustamante, M. J. Bamshad, and H. M. Akey. Evolution and functional impact of rare coding variation from deep sequencing of human exomes. *Science*, 337(64):64–68, 2012.
- [42] The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467:1061–1073, 2010.
- [43] H. Thorvaldsdóttir, J. T. Robinson, and J. P. Mesirov. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Briefings in Bioinformatics*, 14(2), 2013.
- [44] UniProt Consortium. Reorganizing the protein space at the Universal Protein Resource (UniProt). *Nucleic Acids Res.*, 40(Database issue):71–75, Jan 2012.
- [45] Univ. Toronto Biolab. MedSavant variant search engine. <http://medsavant.com>, 2012.
- [46] B. Wang, Q. Michelle, A. Woodruff, and A. Kuchinsky. Guidelines for using multiple views in information visualization. In *Proc. ACM Advanced Visual Interfaces (AVI)*, pages 110–119, 2000.
- [47] J. Wang, L. Kong, G. Gao, and J. Luo. A brief introduction to web-based genome browsers. *Briefings in Bioinformatics*, 14(2), 2013.
- [48] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, second edition, 2004.
- [49] W. Willett, J. Heer, and M. Agrawala. Scented widgets: improving navigation cues with embedded visualizations. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis 2007)*, (13):1129–1136, 2007.