Variant View: Visualizing Sequence Variants in their Gene Context

Joel A. Ferstay, Cydney B. Nielsen, and Tamara Munzner, *Member, IEEE*

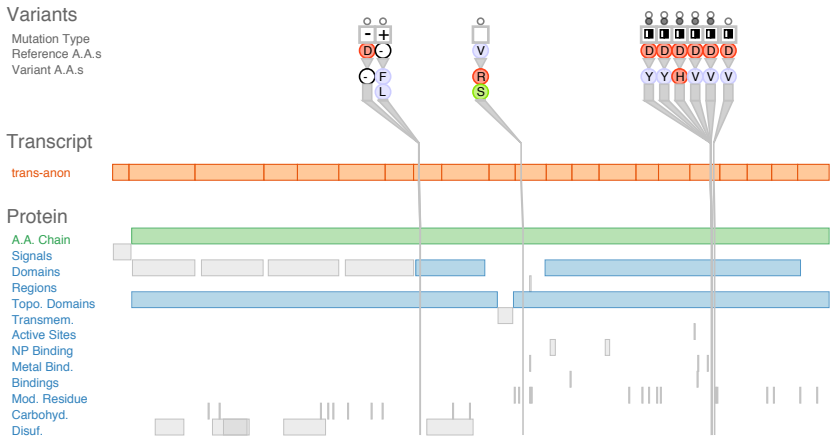

Fig. 1. Sequence variants and their attributes shown in Variant View with respect to biological context annotations at multiple scales. This gene, whose name is anonymized, was identified by analysts as a putative cancer candidate gene through using the tool.

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.

Index Terms—Information visualization, design study, bioinformatics, genetic variants

1 Introduction

The human genome project produced a reference genome for the human species [10], 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 one percent [16]. Differences between an individual person's genome and the reference genome are called *sequence variants*. 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 [3, 7, 11, 32, 34]; 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 [3] or fall short of presenting the full spectrum of biological context needed by the analysts [2, 5].

In this design study, we worked with four variant analysts over a six month period to design and refine Variant View, a tool to accelerate and improve variant analysis, shown in Figure 1. 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.

One contribution of this paper 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,

- *Joel A. Ferstay and Tamara Munzner are with the University of British Columbia. E-mail: {joelaf,tmm}@cs.ubc.ca.*
- *Cydney B. Nielsen is with the University of British Columbia and the BC Cancer Agency. E-mail: cnielsen@bccrc.ca.*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

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 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".

2 Design Process

Our design process followed the collaborative nine-stage design study methodology framework [28], a natural choice since it is the distillation of the experience of conducting over twenty design studies co-authored by one of the authors of this paper. 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. 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 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 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 3.2: 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 [14] 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.

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 3.2. 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 [15], 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.

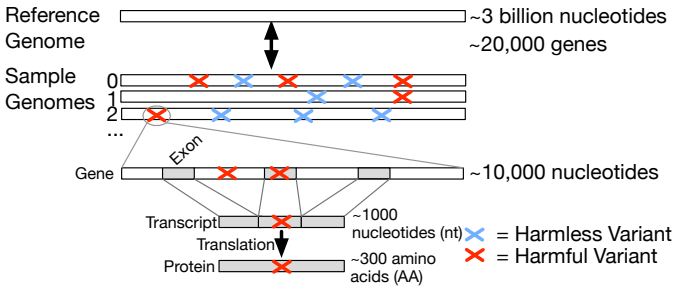

Fig. 2. Biological context in which variants occur. The gene level is specified by genomic coordinates, the exon-containing transcript level is specified by transcript coordinates, and the protein level is specified by protein coordinates.

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 1. Data abstraction. 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.

3 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 [19, 21]. 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.

3.1 Data

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. This imperfect abstraction is actively being augmented by sampling [31] and storing [29] the larger scope of human variability. The genome of an individual is called a *sample*; it typically contains 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 2 shows a diagram of the relevant biological context.

3.1.1 Scales and Coordinate Systems

As with many complex datasets, there is known and relevant structure at multiple scales in genome sequence data (Table 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 that, 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 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.

3.1.2 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 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 classes. 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.

3.1.3 Gene Attributes

Our data abstraction also includes two derived attributes, called *varcount* and *hotspot*, that we calculate for every gene (Table 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.

3.2 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.

3.2.1 Driving Biological Tasks

Discover Genes: The Discover Genes task is to find new genes that are candidates for involvement in the disease of AML 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. They also manually compared the variant against two different databases of known variants [8, 29], to understand whether or not it had already been characterized as being harmful or harmless.

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 AML, in hopes of generating a diagnosis and treatment plan by noting variants similar to a disease population group [30]. 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

	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 2. Concrete questions asked by analysts to infer variant impact, for each of the three identified tasks.

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.

3.2.2 Tasks and Data Questions

Table 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 Section 5, we use these concrete questions to motivate and justify the design decisions we make to construct our visualization solution.

4 Related Work

There are many tools available for visualizing sequence variant data. Some tools target flexibility in the sense of displaying a large number of attributes, some irrelevant to the current study's tasks, and others target particular tasks with limited flexibility. Generally speaking, genome browsers are the most flexible tool [3, 7, 11, 32, 34]; one genome browser in particular, Ensembl [3], 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 [2] and MuSiC [5], but only do so at the gene level.

Genome browsers are the dominant paradigm in sequence visualization today [3, 7, 11, 32, 34]. 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. 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 5.2 as part of our design rationale.

The Ensembl 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 [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 the supplemental materials, typically requires vertical scrolling, particularly to see variants 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.

In contrast to genome browser approaches, there are two recent

tools, cBio [2] and MuSiC [5], that are more tailored to the display of variant attribute and multi-scale annotation information. 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 colocated 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 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.

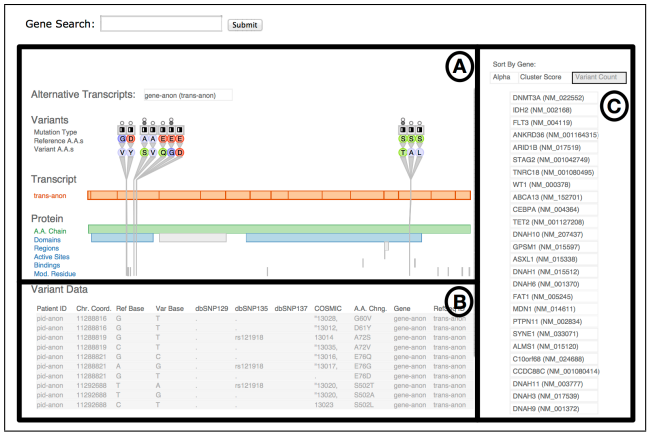

Fig. 3. The Variant View tool, annotated to indicate its three main views. The primary view (A) is the central overview for performing variant impact assessment; the reorderable gene list view (B) can sort genes alphabetically or by derived measures of variant importance; the secondary Variant Data table (C) contains peripheral information.

5 Design Rationale

We now discuss the design decisions for Variant View.

5.1 Core Components: Automation versus Visualization

Figure 3 shows Variant View, with its core interface components labeled. The overall design arose from considering the specific tasks outlined in Section 3.2 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 3.2.1 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 3, 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 3, label A) as possible; to avoid clutter, we show attributes that were deemed by our analysts to be more peripheral to the analysis into the supporting table view (Figure 3, label C). Variant View features bidirectionally linked views [35] 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.

5.2 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. 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 [13, 26]. 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 [9, 25] 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 Section 4, tools like the Ensembl variation image [3] 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 [22, 23]. 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.

5.3 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, the combination of Tables 1 and 2 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.

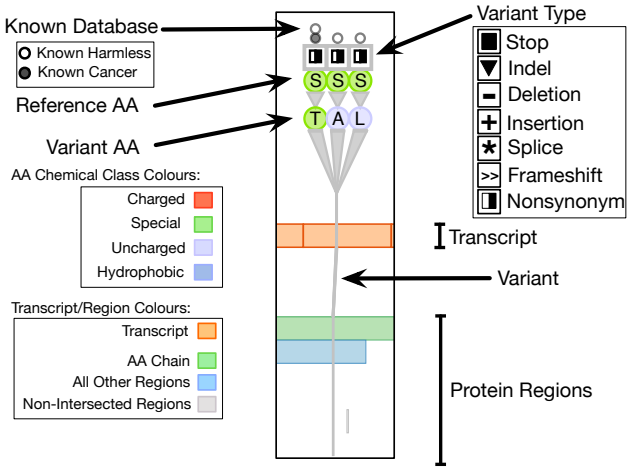

Fig. 4. Variant visual encoding.

5.4 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, 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. 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.

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.

5.5 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 4 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 simultaneously, so a different vertical region is allocated to each. 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 [2, 3, 5]. Moreover, the small size of these marks precludes the effective use of color coding [36] 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). 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. 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 [2, 3, 5].

5.6 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, 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.

5.7 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 5 shows a comparison of variant data for the known gene DNMT3A between the encoding schemes of cBio [2], MuSiC [5], Ensembl variation image [3], and Variant View. Because of the usage barriers described in Section 4, 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 5(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 5(d) shows that the Variant View encoding is both more information-dense and more visually salient than these previous tools.

The Ensembl mockup in Figure 5(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. (The supplementary materials include an annotated complete screenshot of Ensembl showing a different dataset.) 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.

5.8 Implementation

Variant View was implemented using a combination of HTML, CSS, JavaScript, and the JavaScript Data-Driven Documents (D3) library [1]. 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 [33] for protein information and the RefSeq database [27] for exon information.

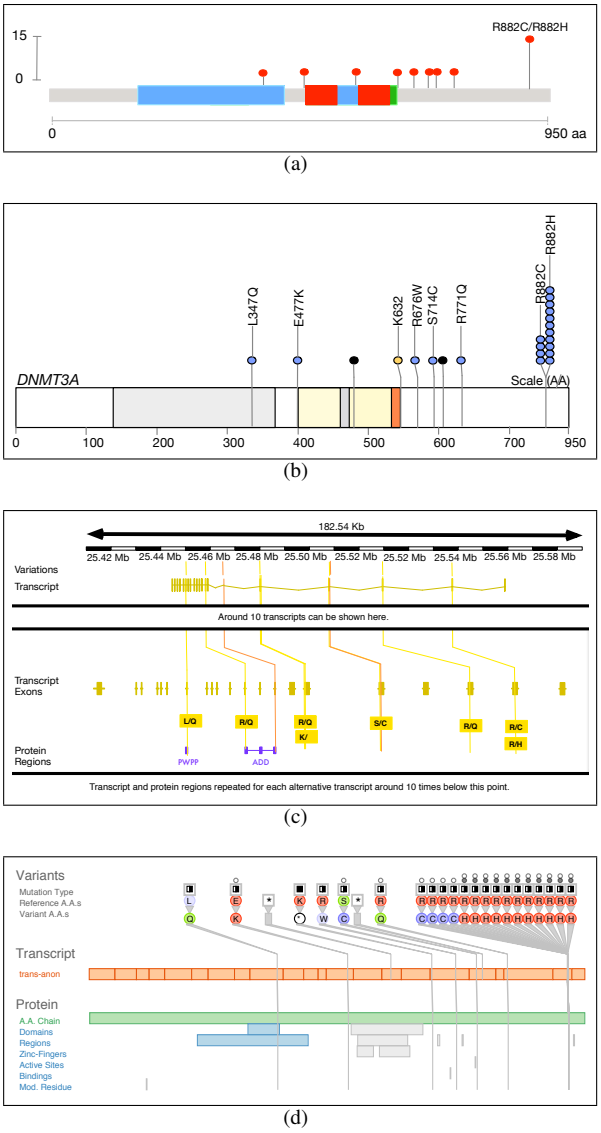

Fig. 5. Comparison of the same variant data between different visual encoding schemes. (a) cBio [2] mockup. (b) MuSiC [5] mockup. (c) Ensembl variation image [3] mockup. (d) Variant View screenshot.

6 Case Studies

We now present three case studies to 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.

6.1 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 3.2.1. 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.

6.1.1 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 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 6(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 6(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 6(a) contain many different types of variants, whereas the cluster in Figure 6(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.

6.1.2 Hypothesis Generation

In addition to retrieving and inspecting known variants in important AML genes, our analysts successfully used Variant View to discover interesting candidate genes. For example, Figure 1 shows one of these genes, and two more are shown in the supplemental materials. 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 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 and the supplementary figures 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.

6.2 Case Study 2: Compare

Analysts A3 and A4 used Variant View for the Compare Patient task, as described in Section 3.2.1. Figure 7 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. 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

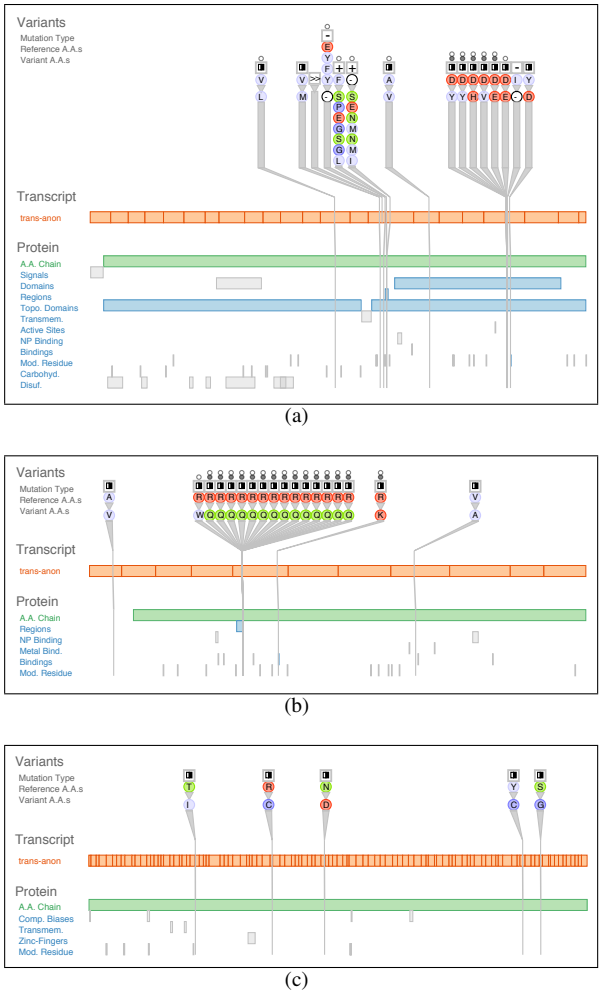

Fig. 6. Variant View allowed analysts to quickly confirm known results: known AML genes could be found near the top of the sorted lists, and the per-gene views clearly and immediately showed tell-tale structure. (a) IDH2. (b) FLT3. (c) Example gene without interesting structure near the list bottom.

analysts remarked on how quickly the tool allowed them to draw these conclusions.

6.3 Case Study 3: Debug Pipeline

The Debug Pipeline task, as discussed in Section 3.2.1, 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 8 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:

> 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.

7 Discussion and Lessons Learned

We discuss the design strategy of "specialize first, generalize later" as a way to tackle biological data visualization challenges. We also

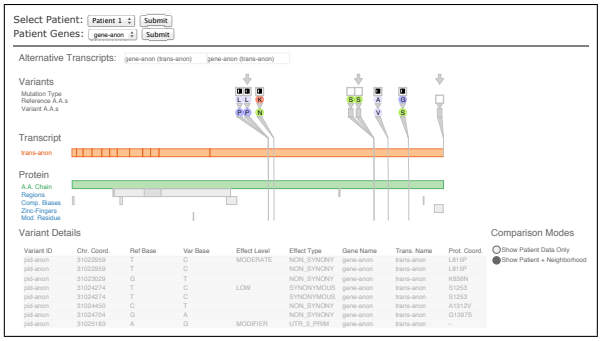

Fig. 7. Comparison of patient data to a known-AML variant database. The immediate neighbors for each variant are shown.

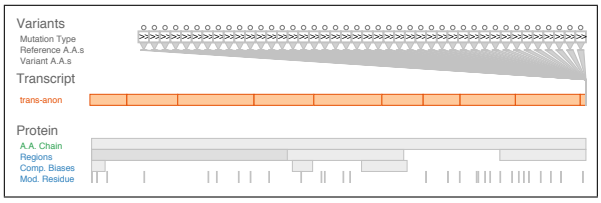

Fig. 8. Debugging the bioinformatics pipeline.

reflect more generally on the visualization design issues, and discuss limitations of the work with an emphasis on the design's scalability.

7.1 Specialize First, Generalize Later

The domain of biology has been a frequent target of design studies in visualization [4, 12, 17, 18, 20, 24]. 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 [21], 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 our experiences in many design studies [24, 28]; 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 adaptions 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.

7.2 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.

7.3 Scalability Limitations and Future Directions

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, but abandoned this effort because of the availability of existing variant management and filtering solutions, such as MedSavant [6], that could be integrated with Variant View as future work. Existing tools, such as cBio [2] and MuSiC [5] have similar display limits, which we estimate at 80 and 60 variants, respectively, onscreen without overlaps. The Ensembl variation image [3] 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. In addition to integrating with data filtering systems, another interesting future direction would be to extend the tool for comparing more than just two groups of patients.

8 Conclusions

In this design study we designed, implemented and deployed a tool for genetic variant impact assessment. It 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, deemphasized, 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.

Acknowledgments

We thank our collaborators at the Genome Sciences Centre: Aly Karsan, Linda Chang, Gerben Duns, Rod Docking, and Simon Chan. Thanks to Matt Brehmer, Jessica Dawson, and Stephen Ingram for feedback on this work. J.A.F. was supported in part by the Vancouver Institute for Visual Analytics (VIVA) and Mitacs, and C.B.N. is a Canadian Institutes of Health Research (CIHR) and Michael Smith Foundation for Health Research (MSFHR) postdoctoral fellow.

REFERENCES

[1] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis 2011)*, 2011.

[2] E. Cerami, J. Gao, U. Dogrusoz, B. E. Gross, S. O. Sumer, B. A. Aksoy, A. Jacobsen, C. J. Byrne, M. L. Heuer, E. L. andYevgeniy 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.

[3] Y. Chen, F. Cunningham, D. Rios, W. McLaren, J. Smith, B. Pritchard, G. Spudich, S. Brent, E. Kulesha, P. Marin-Garcia, D. Smedley, E. Birney, and F. P. Ensembl variation resources. *BMC Genomics*, 11(11), 2010.

[4] 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.

[5] N. Dees, Q. Zhang, C. Kandoth, M. Wendl, W. Schierding, D. Koboldt, T. Mooney, M. Callaway, D. Dooling, E. Mardis, R. Wilson, and L. Ding. Music: Identifying mutational significance in cancer genomes. *Genome Res*, 22:1589–1598, 2012.

[6] M. Fiume, E. J. Smith, A. Brook, and M. Brudno. MedSavant: Visual analytics for genetic variation datasets. BioVis Posters Abstracts, 2012.

[7] M. Fiume, V. Williams, A. Brook, and M. Brudno. Savant: genome browser for high-throughput sequencing data. *Bioinformatics*, 26(16):1938–44, 2010.

[8] 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.

[9] G. W. Furnas. Effective view navigation. In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pages 367–374, 1997.

[10] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.

[11] W. Kent, C. Sugnet, T. Furey, K. Roskin, T. Pringle, A. Zahler, and D. Haussler. The human genome browser at UCSC. *Genome Res.*, 12(6), 2002.

[12] R. Kincaid, A. Ben-Dor, and Z. Yakhini. Exploratory visualization of array-based comparative genomic hybridization. *Information Visualization*, 4(3):176–190, 2005.

[13] 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.

[14] 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.

[15] 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.

[16] M. L. Metzker. Emerging technologies in DNA sequencing. *Genome Research*, 14:1767–1776, 2005.

[17] 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.

[18] M. Meyer, T. Munzner, and H. Pfister. MizBee: A multiscale synteny browser. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2009)*, 15(6), 2009.

[19] 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.

[20] 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.

[21] 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.

[22] C. Nielsen and B. Wong. Points of view: Managing deep data in genome browsers. *Nat Meth*, 9(6):521–521, 2012.

[23] C. Nielsen and B. Wong. Points of view: Representing the genome. *Nat Methods*, 9(5):423–423, May 2012.

[24] 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.

[25] P. Pirolli and S. K. Card. Information foraging. *Psychological Review*, 106(4):643–675, 1999.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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. M. Kang, D. Jordan, S. M. Leal, S. Gabriel, M. J. Rieder, G. Abecasis, D. Altshuler, D. A. Nickerson, E. Boerwinkle, S. Sunyaev, C. D. Bustamante, M. J. Bamshad, and J. M. Akey. Evolution and functional impact of rare coding variation from deep sequencing of human exomes. *Science*, 337(64):64–68, 2012.

[31] The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467:1061–1073, 2010.

[32] 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.

[33] UniProt Consortium. Reorganizing the protein space at the Universal Protein Resource (UniProt). *Nucleic Acids Res*, 40(Database issue):71–75, Jan 2012.

[34] J. Wang, L. Kong, G. Gao, and J. Luo. A brief introduction to web-based genome browsers. *Briefings in Bioinformatics*, 14(2), 2013.

[35] M. Q. Wang Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for using multiple views in information visualization. In *Proc. ACM Advanced Visual Interfaces (AVI)*, pages 110–119, 2000.

[36] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, second edition, 2004.