VisuaLaws: Visualizing Laws over Time

Kimberly Dextras-Romagnino, Yasha Pushak

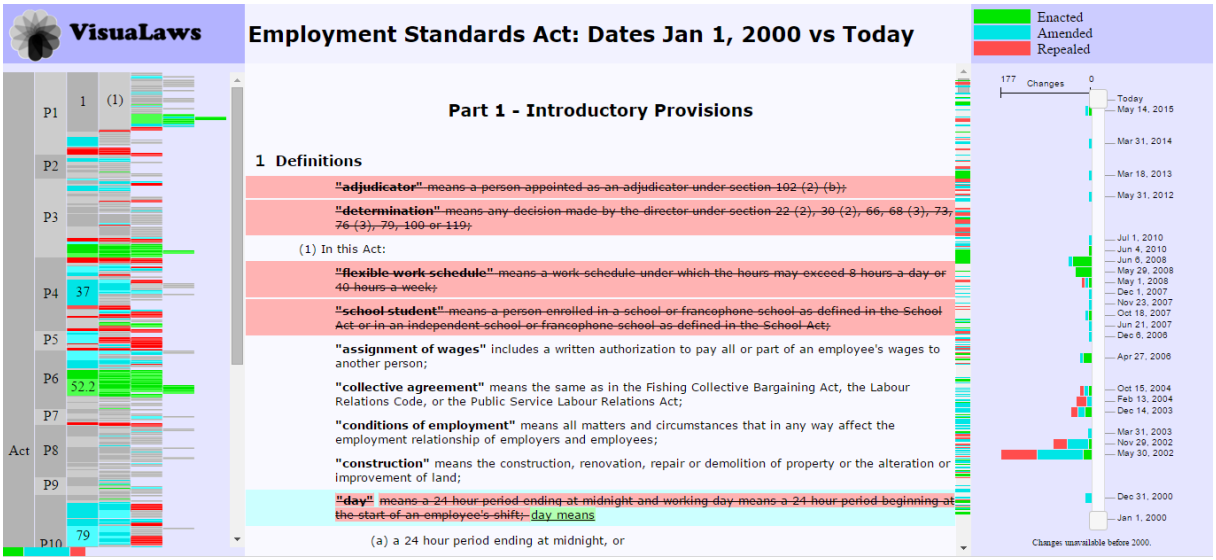


Fig. 1: The Employment Standards Act as seen when first loaded into VisuaLaws.

Abstract—Viewing a law as it was at a particular point in time is an important, yet time consuming task requiring the user to search through a table summary of changes made to the law, comparing them with the current version of the document. Building a mental model of the document at two points in time, and understanding the differences between them is a challenging task for large laws. Understanding the history of how a large law has changed over time is nearly impossible. We introduce VisuaLaws, a prototype web-based tool for visualizing changes to laws over time, which efficiently supports comparing two versions of a law over time and simplifies the challenge of understanding the complex history of changes made in the lifetime of a law.

Index Terms—Law, document history, visualization

1 Introduction

Viewing the current version of a law through the BC Laws website [1] is possible, however they do not support viewing the law as it was at a previous date. Viewing an old version of a law is a common task for lawyers, as criminals are held accountable to the law as of the date of the crime committed. Currently, this task is only supported by manually searching through a table of changes on a separate page which summarizes all of the changes that have been made to the law, containing a reference to the section of the law that was changed, the date the change was made and the text of the law as it was before the change. An even more challenging task when done manually, is to build a mental model of the history of changes made to the entire law. Currently, this task can only be done by comparing the current version of a law to all of the changes, a painstakingly long task. To address these problems we introduce VisuaLaws, a tool which automates the task of assembling the version of a law at any point in time and visually encodes the changes that have been made between two selected dates. Multiple views are used to provide the user with summary and detail views. In one view, the user is able to see a summary of the changes made to the document over time, and select two dates of interest using a slider. A second, dense view provides an overview of the document structure in an icicle plot and summarizes the type of changes made to the document between the two selected dates. Finally, in the main diff view, the user is able to see the text of the law and the changes between the two selected versions. A scented scroll bar on this view and small bar charts added to the icicle plot provide a small visual summary of the changes that are outside of the current view position, providing the user with small visual cues that what is out of sight should not be out of mind.

Section 1.1 introduces the terms and definitions used throughout this paper. In Section 2 we provide a summary of the body of related work in information visualization and file difference tools. We discuss the data and task abstractions we used in Section 3. Section 4 contains a detailed description of our solution and justification of our design choices, as well as an analysis of our solution in the “What? Why? How¿‘ framework by Munzner [17]. In Section 5 we include references to the libraries used for our solution, and a discussion of how we implemented the tool. In Section 6 we discuss how our solution addresses the problem tasks discussed in Section 3.2 and in Section 7 we summarize the strengths and limitations of our solution. Section 8 contains our concluding remarks and possible future directions for VisuaLaws.

1.1 Terms and Definitions

Segment refers to a single block of text within a law. That is, either segment is a part, section, subsection, paragraph, subparagraph, or definition, but does not include their children. For example, a segment referring to a section in a law, only refers to the text immediately following the section number, not the subsections contained within it.

- *Kimberly Dextras-Romagnino is a Masters of Science Student with UBC. E-mail: kdextras@cs.ubc.ca.*
- *Yasha Pushak is a Masters of Science Student with UBC E-mail: ypushak@cs.ubc.ca.*

Segment label is a generic term that refers to the numeric, alphanumeric, or textual label of a segment. In the case of sections and parts, this refers to the section and part number respectively, whereas paragraphs are labeled with letters of the alphabet, and subparagraphs are labeled with roman numerals. In the case of definitions, the segment label refers to the term or terms being defined.

Add is used to refer to both the domain specific terms "add" and "enact".

Edit refers to all of the domain terms "re-enact", "replace", "amend", and "change".

Remove refers to both the domain terms "repealed", and "self-repealed".

Change is used to denote a specific altercation to a law, be it something that was added, edited, or removed.

2 Previous Work

Related work can be broken down into a few related areas: vis tools designed to provide overviews, which in many cases can benefit from concepts for designing dense displays and tools for displaying changes between two versions of a document.

2.1 Overview Tools

Providing an overview of the changes to a law can be considered from two points of view, it can either be useful to see a summary of the changes that of occurred between two dates, or to see a summary of the changes that occurred between multiple years.

2.1.1 Comparing Two Years

Tarantula, by Jones et al. [16], uses a dense display to source code documents where each line of code has two quantitative attributes. Each line of code is given one pixel in height and a bivariate colour map. A similar approach could be used to encode the changes between two documents, where instead of using colour to encode quantitative attributes, colours were selected to encode the categorical data of section added, deleted, or changed. One of the problems we see with adapting Tarantula to our task, is that it will not clearly illustrate the hierarchical structure of the law such as showing the relationship between sections, subsections, paragraphs, and sub-paragraphs.

Schulz et al. [19] discuss several methods for representing the changes between hierarchical documents, including a sunburst, polar Treemap, or InterRing, or a Treemap layout. However, out data set is not well suited to a circular layout as in the first three as the top layer of the tree typically would have many children requiring a large center node which would take up unnecessary space. Tu et al. [20] explored the use of a Treemap to encode changes within a hierarchical document. However, we have opted not to use a Treemap (or a radial layout) as we believe they would be less intuitive to the average user for understanding the structure of a law. Instead this structure could be easily displayed in a much more familiar context using a zoom-able icicle layout also mentioned by Schulz et al.[19], which can then be scented with colours to encode the additional information about added, deleted, and modified sections.

2.1.2 Comparing Multiple Years

While all of the methods discussed in Section 2.1.1 are potential options for providing an overview of changes between two years, they would struggle to show differences between several years, particularly when the same line has changed multiple times in different years. While we believe that it is very important to provide a salient summary of the differences between two years, we also see it necessary to provide an overview of all of the changes to a document that have occurred in the entire timeline.

In contrast to the previous methods discussed, ThemeRiver [15] takes as input a collection of documents and identifies themes over time. These themes are represented stacked area marks whose height encodes the weight of a topic which varies over the x-axis to encode the changes over time. This vis technique was extended by the History Flow Vis idiom presented by Viegas et al. [21],to visualize changes made to Wikipedia articles over time. The horizontal axis was again used to represent time with the vertical axis encoding the height of the document. Each line was then colour coded to reflect the author. This view allowed users to see how the document structure grew, shrank, and shifted over time.

TreeVersity2 [14] is a tool that visualizes changes in large hierarchical data sets over time similar to what we are trying to do. It contains an overview line graph showing the aggregated changes in a given time interval as well as another view showing stacked bar graphs (centered in the middle) representing the aggregated changes between any two given years. There is also the option of breaking the view into smaller components to gather more detailed information. Though we won't be using the same idioms due to differences in structure of our data set, we plan to use the same idea of combining general overview of the changes in law over time as well the comparison between two dates and finally, having the ability to drill down to see further details.

2.2 Dense Displays

Displaying a summary of an entire law and all of its changes on the screen at one time is a challenge requiring the use of a dense display. Fekete et al. [12] faced a similar challenge in an attempt to visualize millions of items at one time. In order to distinguish the boundaries between their rectangular objects scattered throughout the page, they used gradient colouring to allow users to distinguish between adjacent objects which would have otherwise blended together due to their similar colouring.

Munzner et al. [18] guarantee visibility of the important, highlighted elements in the fixed screen display by specifying a minimum height of 1 pixel, despite that these objects would normally fill sub-pixel height. While this is a potential option for dense displays, it is not as applicable to our solution because we typically need to highlight thousands of elements, so providing 1 pixel height for each would still require more screen real estate than available.

2.3 File Difference Tools

File difference tools are used to compare changes between two text files. In some cases [3, 2], two versions of a file are given as input to the tool, which then displays each side-by-side using colour-coded mark-ups such as green highlights in one file, to indicate an added section of text, and red highlights in the other, to indicate a deleted section of text. Other tools, such as the "Track Changes" feature in Microsoft Word 2013 [7], only show the document in one view, and embed the changes made to it, i.e., inserted and deleted text, within the single view. In this case, changes in text are encoded with a font colour change, with deleted text represented using a strike-out mark and added text underlined. One clear advantage of this representation is that reduces the amount of screen space needed for the vis idiom. On the other hand, when extensive changes are made to a document, this technique may become more cluttered, and the user may have more difficulty in piecing together the individual changes to build a mental model of each document version. When two side-by-side views are used to display each version of the document, it is easy to see what is contained in each version of the document, however unnecessary additional screen space is needed to encode regions of the documents that are identical in each document.

3 Data and Task Abstractions

The data and tasks associated with the tool are specific to the domain of law. They were both abstracted into concepts that can be more easily analyzed in order to develop the best design solution.

3.1 Data

The data comes from the laws in the Statutes and Regulations section of the British Columbia Laws (BC Laws) website. The information for each law is contained in two separate XML files, the law XML and the change XML. We transformed the domain-specific data into a form that was easy to work with.

System	**VisuaLaws**
What: Data	Two XML files: one represent the law one representing the changes over time
What: Derived	Two-dimensional table with categorical attributes

Table 1: The What analysis of the VisuaLaws tool

System	**VisuaLaws**
Why: Tasks	Looking up, locating, browsing, and exploring values with specific attributes, identifying outliers or points of interest, summarizing data

Table 2: The Why analysis of the VisuaLaws tool

3.1.1 Data: Domain

The law XML contains the most recent version of the law. The laws have a nested structure where the segments are labeled in the following descending hierarchical order: parts, divisions, sections, subsections, paragraphs, and subparagraphs. Definitions can also be included under either sections or subsections. Each segment is identified by a unique id. The change XML contains information on how the law has changed over time via a list of changes. Each change includes the date the change was applied, the type of change, the segment affected and what that segment looked like immediately before the change occurred. Types of changes include: added, enacted, re-enacted, amended, replaced, changed, self-repealed, and repealed.

There are 736 laws [2] included in Statues and Regulations of British Columbia which range in size from having 3 sections to 1040 sections. We chose a small subset of 10 laws, including the smallest and the biggest, to visualize as we believed they represented the scope of our project. The number of changes corresponding to any given law in this subset range from 1 to 1115.

3.1.2 Data: Abstraction

We transformed the information in the law XML into a table where each row represents a single segment of the law. We essentially flattened the entire law by ignoring nested relationships between segments. Each row has the following categorical attributes: id, segment label, type, and text where the id attribute is unique for every segment, type refers to the type of segment it is, and text refers to only the text included in that segment; it does not include the text of its children. We also generated an Id look up table from this XML to help easily identify the ids of segments whose ids are not provided. The keys of the look up table are the concatenated string of segment labels of the parent segments leading up to a specific segment and the value is the corresponding unique id of that segment.

Similarly, we also transformed the change XML data into a table where each row represents a particular date that a change occurred. The attributes for each row include a date and a list of changes that occurred on that date. The date attribute is both categorical and ordered from least recent to most recent. The list of changes is essentially its own table where every row refers to a change object which represents a specific segment that was changed. Many change objects can be generated from a single change element in the XML file as a change object is created for every affected segment including the children of the primary affected segment. Every change object has the following categorical attributes: id, segment label, type, type of change, and text. Type, segment label, and id refer to the information about the segment being changed. The change XML does not initially include ids and we therefore had to retrieve them from the Id lookup table generated from the law XML. If an id was not found, a unique id was generated and added to the lookup table for future reference. We aggregated the type of changes into three main categories: added, removed, and edited. The text attribute contains the text of the segment before the change was implemented. If no text is given, it is set to an empty string.

3.2 Task

Given that the information is distributed between two separate XML files, it makes it extremely difficult to actually see how a law evolved over time. The goal of the tasks our tool supports is to address these problems. We defined our tasks based on the two different types of users who would use our visualization tool: legislators and academics studying law.

3.2.1 Task: Domain

An accused is only held accountable to the version of the law as of the date the crime was committed. Therefore, a user who is a legislator will want to be able to see what the law looked like on the date of their client's crime. They may also want to compare that version of the law with a version from another year. This involves viewing what a law looks like at any given date as well as being able to view the Previously, this task was very difficult given only the most recent version of the law is available on the BC Laws website. The legislator will also want to immediately identify and investigate the dates where the most changes occurred. The legislator's tasks are more specific as they generally have an idea of what they are looking for.

On the other hand, the academic might be more interested in understanding the evolution of a specific law over time. This may involve finding out what specific segments of a law were changed the most often throughout the years or what years resulted in the most amount of segments added or repealed. The academic will want an overview of the changes to get a general idea of how each law changed. This information can then be used to draw conclusions or enforce a point in an academic paper regarding the history of the law.

3.2.2 Task: Abstraction

The general tasks supported by our tool that come directly from the legislator use case fall under the categories of Search and Query. A legislator will generally have an idea of what information they are searching for whether it be a known target, such as a specific type of change, particular segment of interest, or certain version of the law, or a known location, such as a specific date of interest. If a legislator knows their target, they may still be aware or unaware of the location of the target as in what date it occurred. If the location is unknown, the user should be able to locate the date where the specific change occurred. If the location is known, a user must be able to look up a specific change or see the version of the law on that specific date. If the location or date of interest is known but the specific target is unknown, the user must be able to browse through the changes in the date to find the point of interest, When querying, the user should be able to easily compare the differences between two specifics dates of a law. They should also be able to query further to identify the exact changes, such as the actual difference between the text of the two versions of the law, that occurred between the two dates.

The general tasks supported by our tool that relate more to the academic use case fall under the categories of Search, Query, and Analyze. In general, the academic will not have a specific target or location in mind when searching through the changes of a particular law. They should be able to explore through the different dates and examine the different types of changes that occurred to the various segments of the law. In terms of querying, the user should be able to get an overall summary of all the relevant data including an overview of all the changes that occurred over time and what specific segments of the law were affected by these changes between any two dates. In terms of analyzing the data, the user should be able to discover any points of interest or outliers such as particular segments of the law that were changed more than once throughout the years, dates where significant changes occurred, or segments of the law that were repealed and added. The user should also enjoy using the tool as a lot of interesting information about laws can arise from interacting with the tool.

4 Solution

In Figure 1 we present the five views comprising VisuaLaws. In the middle, is the main diff view, which allows the user to see the text of the law and the changes made between two dates.

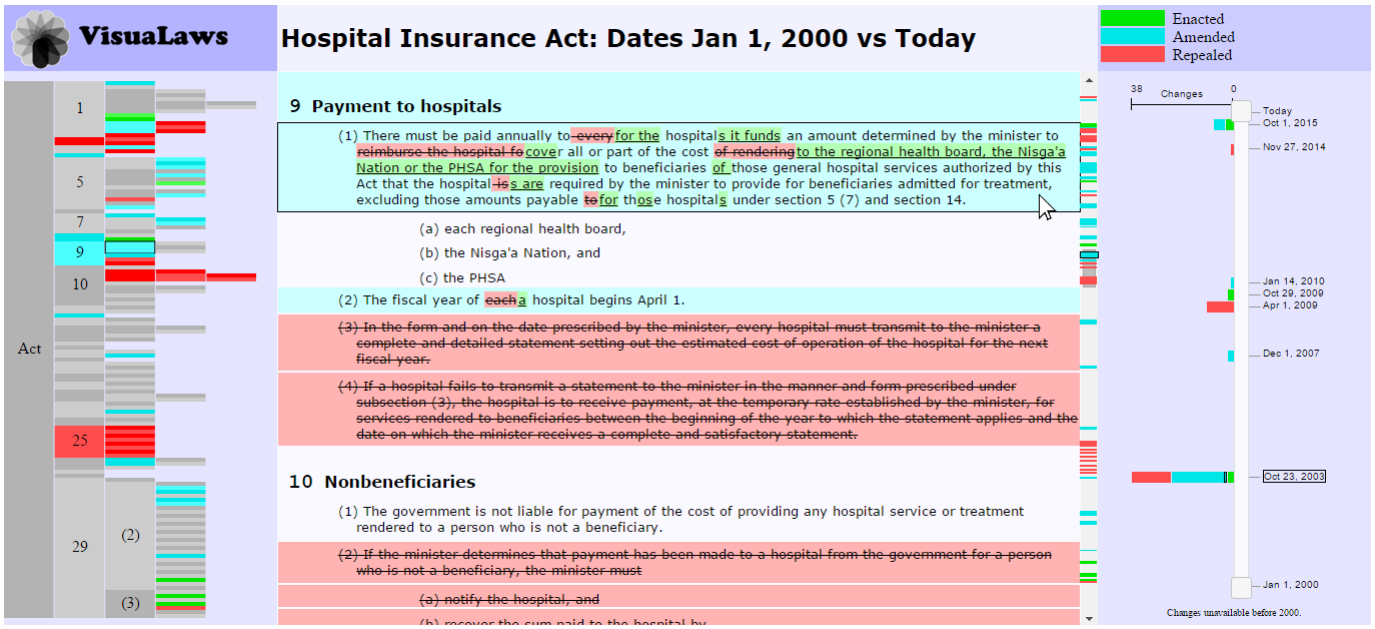


Fig. 2: An example of linked highlighting across the different views. When the user hovers over the segment in the main diff view the corresponding elements in each other view highlight, allowing the user to identify where they are in the document structure view and the scented scroll bar view, and when the highlighted segment changed in the timeline view.

Throughout each view, we use the categorical hues green, cyan, and red to represented segments of the law that have been added, edited, and removed respectively, which is summarized in the legend in the top left corner. Green and red were used to represent added and removed because of their strong association with the concepts, as evidenced by their wide spread usage in other file difference tools. We selected the colour cyan to represented edited segments, because it has a similar pop-out effect as the green colour due to its similar perceived luminance, rather than the orange or yellow seen in some other tools. The colour red, however, posed some measure of difficultly in initial versions of the tool, as it is perceived as a much darker colour and so had a much stronger pop-out effect. To help overcome this challenge, we used a less saturated red, so that the perceived luminance more closely matches those of the cyan and green.

When a user hovers over an object in one view, it, along with the corresponding object or objects in the other views, become highlighted using a 1 pixel black border, allowing the user to quickly identify the relationship between objects in each view, as seen in Figure 2. In most views, when an object is highlighted, the border is drawn over top of the existing space taken up by the object, so as to avoid increasing the height of the object. This constant height and width is maintained so that objects coming after the highlighted one(s) do not jump up and down the page, a motion which would distract from the user's ability to identify which items have been highlighted.

4.1 Main Diff View

In the main diff view, the user is able to see the text of the law, and the changes between two dates. We opted to use a single view for this component that contains both version of the document, then and now, rather than splitting them each into their own views. While this decision does require a bit more effort on the part of the user to mentally build a model of each version of the law, it requires half of the screen space required to split into multiple views, freeing up valuable space for the other views to utilize.

Segments of the law that have been added, edited, and removed use relatively unsaturated versions of each of the categorical colours as their highlight background colours to indicate their category without overstimulating the user by displaying large, bright patches of colour. As seen in Figure 3, we have followed a common convention among file difference tools and redundantly encoded added and removed segments using an underline and strike-out text decorations, which can help colour-deficient users distinguish between the two categories. The semantics of an edited segment, often, but not always, mean that part of the segment's text has been added, and part has been removed. We have encoded these edits to the text using a similar markup to the added and removed sections for visual consistency. Determining which portions of text in an edited section have been added and removed is done using a generic text difference tool.

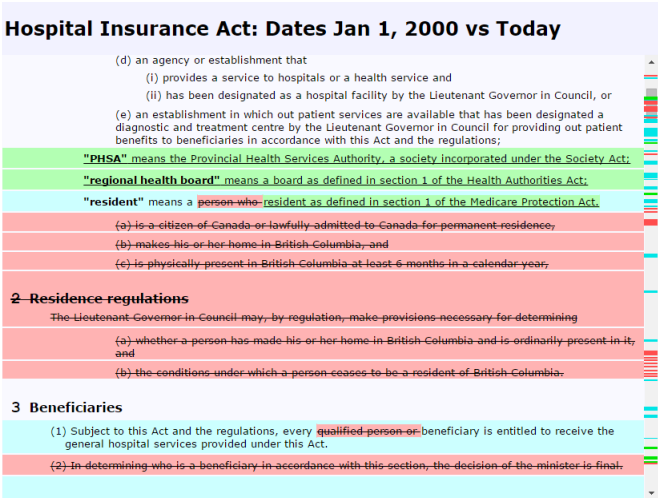


Fig. 3: An example of the three colours, green, cyan, and red, used to encode the categories added, edited, and removed, respectively. Edited segments may also contain portions of text that have been added and removed within them. These portions of text, and the added and removed sections, have underline and strike-out text decorations to redundantly encode their categories.

Linked highlighting is also triggered by this view when the user is scrolling up or down. In this case, all of the objects that are currently displayed within the view are highlighted in this view, and the corresponding objects are highlighted in each other view, as see in Figure 4.

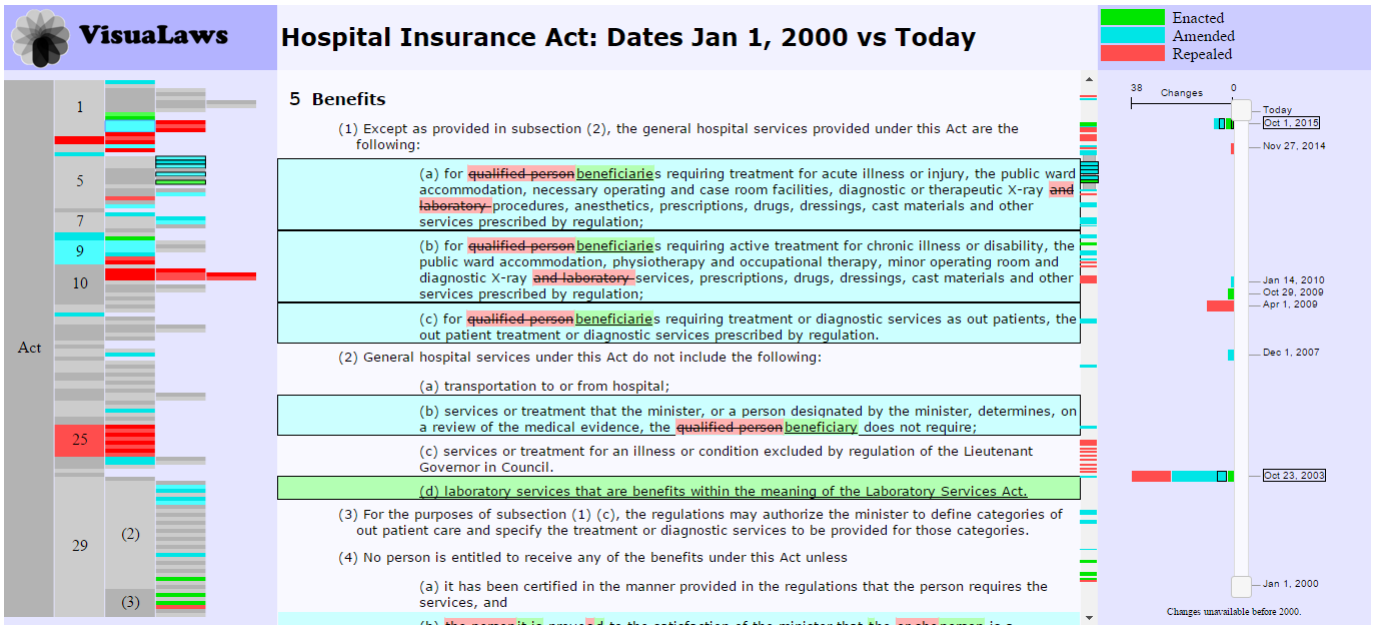


Fig. 4: An example highlighting that occurs as the user scrolls. The five segments entirely in the current scroll frame of the page are highlighted, along with their corresponding scented scroll elements. We see that these segments are positioned in Section 5 by examining the document structure view, and that changes occurred to these segments on October 1, 2015 and Oct 23, 2003. We also see in this Figure how the size of the scented scroll bar elements car vary. For the small segments, they are displayed with as little as 1 pixel, while the larger ones may take up to 5 pxiels.

View	**Main Diff View**
How: Encode	categorical hues
How: Facet	Overview and detail, multiple linked views
How: Manipulate	Linked highlighting with borders, linked navigation

Table 3: The How analysis of the Main Diff View

View	**Scented Scroll Bar View**
How: Encode	categorical hues, redundant underline and strikeout
How: Facet	Overview and detail, multiple linked views
How: Manipulate	Linked highlighting with borders, linked navigation

Table 4: The How analysis of the Scented Scroll Bar View

This helps the user identify where they are in the law when browsing, and helps the user identify which dates contain changes to the currently viewed segments of the law. When the user stops scrolling, the current objects remain highlighted until the user hovers over a new element, at which point the new element is highlighted in each view instead.

4.2 Scented Scroll Bar View

Given the size of most laws, it is impossible to display the entire law at one time while using a readable font size. To help the user navigate and identify regions of interest within the law, we have overlaid the scroll bar of the main diff view with small rectangles which correspond to the changed sections within the document. The color of each rectangle uses the three colours to represent the category of the corresponding segment, and its position on the scroll bar reflects the position of the segment in the law. This view requires minimal additional space while allowing the user with the ability to quickly obtain a summary of the changes and their distribution throughout the law.

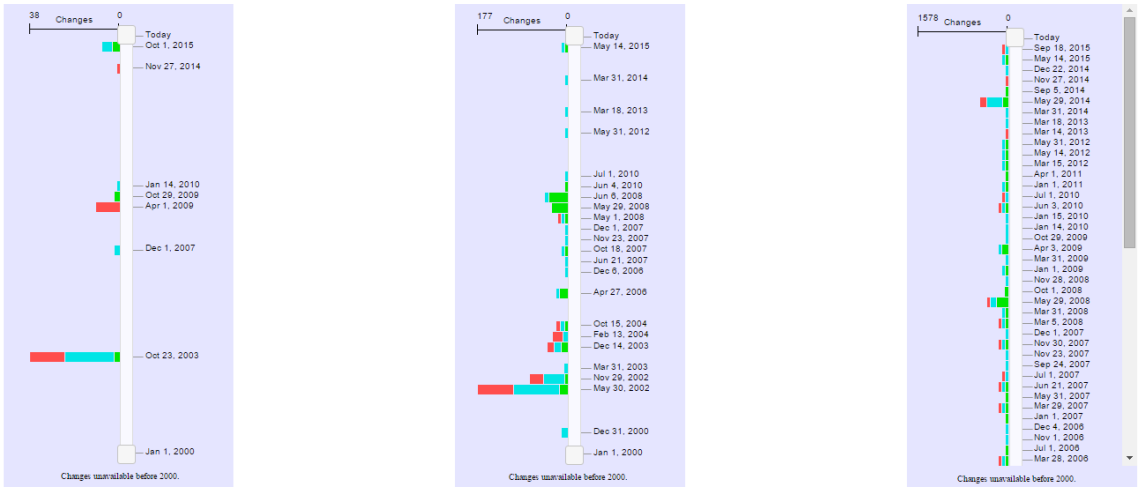
The default height of each rectangle is 5 pixels, however to help avoid occlusion in large laws, their height is decreased to a minimum of 1 pixel. An example of these different heights can be seen in the Hospital Insurance Act in Figure 4, where the smallest element has a height of 1 pixel near the bottom in green, and below it there are others that are taller. Unlike other objects, when the scroll elements are highlighted, the border is drawn around the rectangle, so that the category of the rectangle can still be identified by the colour within the border. Since the position of the rectangles is fixed, rather than based on the elements coming before them in the document, this does not cause the problem of items jumping up and down the page. When two rectangles do occlude each other, the highlighted one is brought to the top layer of the page, to ensure visibility.

The rectangles in the scented scroll bar support linked navigation, allowing the user to click on a rectangle of interest, causing the main diff view to automatically scroll to the corresponding segment.

4.3 Timeline View

On the far right, is the timeline view, which both provides a summary of the changes made to the law over time, and allows the user to select two dates for comparison in the other views. On the right of the slider are the dates at which changes to the law occurred, and on the left are stacked bar charts, which summarize the type and amount of changes made at each date. At the top of the slider, is an axis with a label indicating the largest number of changes made on any date. Since change data is unavailable before the year 2000, the timeline slider stops on January 1, 2000, and small text appears below the slider to indicate that additional data is unavailable.

The slider is positioned vertically so that text labels can be drawn beside it without the need to rotate the text. Since standard fonts are designed for maximum salience with minimum pixels at a standard orientation, this allows for more dates to be fit on the screen at one time. The tick marks for the dates are positioned along the slider following two rules to guarantee optimal placement. The first rule is a minimum spacing between two adjacent tick marks. This is guaranteed by only allowing tick marks at evenly spaced intervals such that two adjacent labels will not not occlude each other. If there is not



(a) An example of the timeline slider with lots of space. We see that the dates labels are positions according to when they occurred, using a quantitative alignment.

(b) Here the timeline slider labels are positioned using a compromise between a quantitative and ordinal alignment. We see dates Dec 6, 2006 to July 1, 2010 are all positioned using an ordinal alignment, while the dates May 31, 2012 through Today, have enough room to be spaced apart, approximating their quantitative alignment positions.

(c) An example of the timeline slider with no free space. We see that the dates labels are positioned ordinally, and a scroll bar allows the user to scroll.

Fig. 5: Three examples of the timeline view with different numbers of date labels.

enough space to display all of the dates along the slider without occlusion, then a scroll bar is added to allow more space. If there is enough space, then each label's position is determined based on the derived, quantitative attribute of how far it is between the first and last date on the slider in time. This quantitative attribute is then rounded to the nearest available tick position. If two dates round to the same tick position, then one or the other is either moved up or down to avoid collisions. This collision avoidance may possibly cause a ripple affect as other labels are displaced to allow room in regions where many changes occurred at similar dates.

While this layout does not strictly preserve the quantitative attribute of position in time, it does provide an adequate compromise between the two costs of allowing the user to quickly identify regions of time of little or no activity, and being able to fit all of the elements on the page at once. In effect, this algorithm treats the date labels as quantitative attributes when space is available, and otherwise as ordinal attributes in an attempt to display all of the information on the page at once. An example of the timeline slider where the dates are treated as quantitative data can be seen in Figure 5a, as the available space decreases, we can see an example of a compromise in Figure 5b where some of the dates are beginning to be grouped together at equal intervals where before they had non-uniform spacing. Finally in Figure 5c we can see the same example when there is not enough space to display all dates at once and a scroll bar is needed.

In a prototype version of VisuaLaws, the stacked bar charts were positioned just underneath the date labels to indicate to the user that the changes represented the differences between that date, and the preceding date. They were moved to the opposite side of the slider in the final version to increase the scalability of the system, allowing for more dates to be positioned on the page before a scroll bar is required. Though this does increase the scalability of the system, further user studies would be needed to determine whether this change detracts from the overall salience of the slider.

The width of each bar in the bar chart encodes the number of changes that occurred on that date, relative to the maximum number of changes that occurred on any individual date for that law. A minimum width of 3 pixels is used to guarantee visibility of each bar when they are highlighted, so that the 1 pixel border on each side does not obscure the 1 remaining pixel of colour in the center. In this case, unlike the rectangles in the scented scroll bar, increasing the size of the bars on highlighting would be inappropriate, since the width is used to encode the number of changes. While the minimum width of 3 pixels does mean that the width of the bar is initially non-linear, it is worth the cost as it also allows the user to easily identify the colour of the bar.

Each bar has 1 pixel of spacing between adjacent bars to allow the user to easily distinguish between the blue and cyan colours for bars with a small width. While this design choice also affects the perceived combined magnitude of the stack bars, we do not believe this to be a serious flaw, as the intent of the bar chart is to provide a quick summary of the changes on the date to allow the user to quickly identify dates of interest, rather than providing a detailed view.

The semantics of the linked highlighting in this view is slightly different than the others, since each date represents a collection of changes. When the user hovers over either one of the stacked bars, or the date, all of the changes corresponding to that date are highlighted in the other views. In addition, a box is drawn both around the stacked bars, and the date labels. When the user selects an element in another view to be highlighted, the date label is highlighted, and the corresponding fraction of the stacked bars are highlighted. That is, if a user is highlighting half of the sections that were added on a particular date, then the black outline would highlight only half of the green bar on that date. The date label highlights completely no matter how much of the bars are highlighted to help the user easily identify which dates are being highlighted when only a small fraction of their changes are being highlighted.

Linked navigation was not originally a planned feature for the timeline slider, however when the first prototype was implemented, users were observed attempting to click on the dates to navigate to the corresponding sections in the document. As a result, we included the ability to cycle through each segment in the document by clicking on a date multiple times. While this feature is likely relatively ineffective for dates with lots of changes, it can be useful for dates with only a few changes.

4.4 Document Structure View

On the far left, is the document structure view, which summarizes the structure of the law using an icicle plot with each segment rep-

View	Timeline View
How: Encode	categorical hues, stacked bar chart
How: Facet	Multiple linked views
How: Manipulate	Select two dates

Table 5: The How analysis of the Timeline View

resented as a rectangular area mark. Echoing the encoding in other views, the colour of the area marks represents the categorical information of added, edited, or removed, as seen in Figure 6. Segment labels are drawn over top of the segments when the area mark contains enough space. We decided to use a vertical icicle plot rather than a horizontal one to echo the vertical structure of the law.

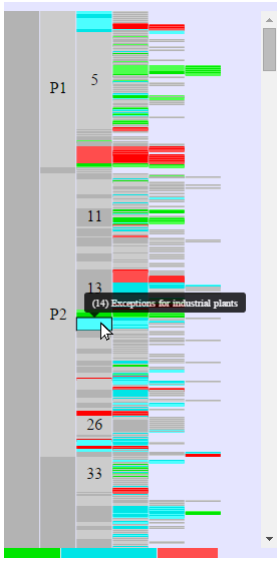


Fig. 6: An example of the document structure view where the highlighting does not obscure the colour of the segment.

Each segment is guaranteed visibility with a minimum height of 1 pixel. In order to distinguish between each adjacent segment, we alternated the luminance of adjacent segments. However, since there are not enough pixels on the screen to display every segment of a law for large laws, a scroll bar is added to avoid sub-pixel heights. A possible alternative to a scroll bar, as in TreeJuxtaposer by Munzner et al. [18], would have been to use sub-pixel widths only for segments without changes while guaranteeing a minimum of 1 pixel for segments that had changed. We opted not to use such a strategy, however, because from our limited selection of large laws, we observed that in most cases, even displaying only the changed segments at 1 pixel each, omitting the rest, would require more space than available in standard displays. Given this, we concluded the only viable option would be to include a scroll bar.

In this view, the highlighting does obscure the colour of the segment when the segment has less than 3 pixels in height, as the border is again drawn within the existing size of the element. While ideally this view would have benefited from the style of highlighting used in the scented scroll bar, this was not implemented due to time constraints. Figure 6 contains an example of a highlighted segment which does not obscure the colour of the segment, while Figure 7 contains an example where the colour is no longer visible.

In addition to highlighting the segment on hover, in this view a title text pops up on however which displays the text within that section of the law. This allows the user to drill down to see more information without requiring them to navigate to that segment in the main diff view. An example of the title text is displayed in Figure 6. We chose to place the tool tips above the selected segment to ensure the the children

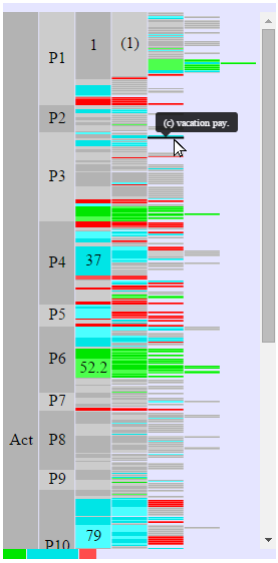


Fig. 7: An example of the document structure view where the highlighting does not obscure the colour of the segment.

of the current segment would remain visible, as it is likely that the user will also be interested in seeing the children of the current segment.

Geometric zooming is performed when the user double clicks on a segment, so that the height of that segment now takes as many pixels as were used in the original icicle plot, with its children increasing in size to match. Should any of the area marks become large enough to support the text labels, they are now drawn as well. An example of a zoomed icicle plot can be seen in Figure 8.

View	Document Structure View
How: Encode	categorical hues, icicle plot, dense display
How: Facet	Multiple linked views
How: Manipulate	Navigate: geometric pan and zoom navigate: scroll
How: Reduce	focus + context

Table 6: The How analysis of the Document Structure View

4.5 Summary View

In order to help overcome the challenge that what is out of sight is out of mind, we added the summary view, to serve as small indications of the changes not displayed in the document structure view. In this case, we again employed the idiom of stacked bar charts to encode the fraction of changes that are above and below the current scroll position of the document structure view. By placing these bars directly above and below the document structure view, the user is able to quickly see as they scroll up and down, or zoom in and out, how many changes are not in view. The bar heights are normalized to the total number of changes that are currently selected by the timeline slider view. That is, if the timeline selection range is increased, this may also affect the heights of the bars in this view, either by shrinking them as more changes are added in the view, or by increasing them as more changes are added outside of the view. Figure 6 contains an example of the summary view which indicates that there are no changes above the current scroll position of the document structure view, while Figure 8 indicates that there are many out of view.

Due to time constraints, linked highlighting was not implemented for this view, though we do believe that this would be a valuable feature for future versions of the tool.

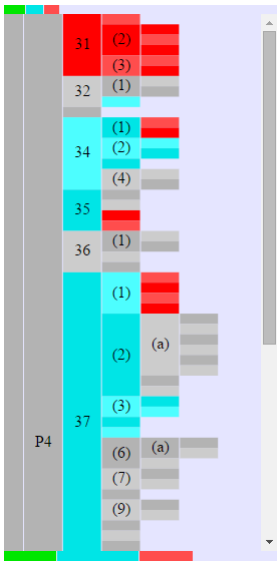


Fig. 8: When the user double clicks on a segment the document structure view zooms in to that segment to allow for a more detailed view.

View	**Summary View**
How: Encode	categorical hues, stacked bar charts

Table 7: The How analysis of the Summary View

5 Implementation

VisuaLaws is implemented entirely as a front-end web-based tool in Javascript. The page elements are defined using a mix of HTML 5 and SVG, using custom defined css to style everything but the elements we used from existing libraries, though many of these required modification as well. In particular, two libraries have been used throughout VisuaLaws: d3.js [10], and jQuery [11].

5.1 Data Parsing

The main goal of parsing the data was to transform the information in both the law XML and the change XML into a format that could be easily inputted into the visual design elements. This task turned out to be significantly difficult. In order to be able to begin coding the visual components, we manually parsed a medium size law before beginning on the real parser. We chose to implement the parser in Javascript because it allows for easy integration of our prototype. Since the format the data needed to be in was similar to the format of JSON objects, we decided to use an online script that automatically parses the XML information into a nested JSON object. However, the script turned out to be ineffective, as JSON objects group similar children together in arrays as opposed to keeping their relative order under the parent object which was essentially in this situation. Modifying the existing script would have taken too much time and effort so we decided to write an individual parser for both XML files from scratch. Both parsers use the library jQuery [11] to retrieve information from the XML elements and use Javascript to parse the data.

Parsing the law XML was relatively straight forward as all the elements contained in the XML file are nicely nested and contain all the necessary information. The algorithm of the parser involves iterating through the nested XML elements and storing the information of each element referring to an individual segment of the law into a corresponding object and then adding that object to an array. When the parser finishes traversing the entire law XML file, the array represents the final flattened version of the law.

Parsing the change XML is not as simple. The goal of the parser is to create a change object that represents every segment that was changed throughout the years for a particular law. Each change object contains an id and information that can easily be matched and compared with an object in the flattened array described above. The information in the change XML is given as a list of changes where each change element contains a change note which is a text block that describes what type of change occurred and lists the segments that were affected, followed by unnested XML elements which represent the text of segments of the law before the change happened. An example of a change note is: "Section 15(1)(d), (2) and (3)(b) BEFORE amended by 2010-6-42 effective July 1, 2010" which is followed by three XML elements that represent the corresponding subsection and paragraphs. The algorithm of the parser involves using regular expressions to extract the relevant text in between brackets in the change note. Taking into consideration the order of the extracted texts as well as whether it is a number, character, or word, the goal is to determine the segment labels of the changed segments as well as the segment labels of all parent segments leading up to that segment. The segment labels leading up to a changed segment are then concatenated together and used to look up the corresponding id of the changed segment in the Id look up table. The following XML elements are then parsed to extract the changed text. The XML elements are not nested so the parser keeps track of the type of segment as well as the segment label of all previous XML elements in order to be able to find the corresponding id of each element. Each XML element that refers to a different type of segment is also parsed differently as they all refer to the needed information with unique XML tags.

The parser handles certain types of changes uniquely. Most segments that were repealed do not appear in the most recent version of the law and therefore do not have a corresponding id or an object in the flattened array representing the law. In this case, the parser creates a new segment object as well as a unique id and inserts the object into the appropriate place in the flattened array so it can be referenced later. Similarly, segments that were added are handled in a special way. All sub-segments of the 'added' segment are found by looping through the flattened array and a change object with the type of change indicated as 'added' is generated for each of them as well.

Due to time constraints, the parser is not implemented to its fullest functionality. It also has a couple bugs and the performance is not optimal. However, it works well enough to show the evolution of the subset of laws chosen for our prototype. Exploring other languages other than Javascript to parse the files might be an option in future work to improve performance.

5.2 Visual Components

The full visual component system is broken down into around a dozen different files that we implemented, though they can be best summarized by talking individually about each view. Each view has their own definitions for the onmouseover, onmouseout, and onclick events, to handle linked highlighting and navigation. The spinner used when loading a law was generated using an open source plugin [8].

5.2.1 Main Diff View

The bulk of the work required for the Main Diff View is done by two functions, bindNoDiff, and bindWithDiff. The first function, bindNoDiff, begins with the most recent version of the law, and iteratively applies the changes to it moving backwards in time until it has created the document as of the first date selected by the user. These changes are applied by binding the data using d3 to divs on the page. This is done without tracking the differences that are being applied. Once this document version has been constructed, the second function, bindWithDiff, iteratively applies the changes again working backwards in time until the second date selected by the user has been reached. As these changes are applied, we keep track of what the differences are between the current date, and the first date selected by the user. When a segment that has already had a tracked change is found to have a second change applied to it, we simply throw out the first tracked change and replace it with the second. This is because the current version of the tool does not support quick identification of all changes, though future versions of the tool could instead use another form of visual encoding to express the intermediate changes.

The data format passed in to this part of the code already contains a categorical variable indicating whether or not a segment was added, edited, or removed. When an added segment is found and when differences are being tracked, the segment is simply highlighted in green, and when a removed segment is found, it is simply highlighted in red. When an edited segment is found, the segment is highlighted in cyan, and a text difference tool by Fraser [13] is used to determine which portions of the text need to be highlighted in green and red.

5.2.2 Scented Scroll Bar View

The scented scroll bar is initially set as an empty collection of elements. Before the bindNoDiff function is called, any existing scroll elements are set to be invisible. During the bindWithDiff function in the main diff view's construction, a call is made for each segment to the colourScroll function. This function checks to see if a scroll element with the id corresponding to the law segment already exists. If not, one is created with a height and position corresponding to the location and height of the segment in the page. The class of the scroll element is then set using d3.js to either added, edited, or removed. When the page is resized, the position of each scroll element must be recomputed to stay on top of the scroll bar.

5.2.3 Timeline View

The timeline view uses an extended version [5] of the jQuery-ui slider [4]. This extended slider allowed for tick marks and labels to be placed on the original jQuery-ui slider. However, our solution required the labels to be positioned according to the quantitative attribute of the date. In order to support this using the existing implementation, we first calculate the maximum number of tick marks possible before we will need to add a scroll bar. If the number of dates is less than this, then we simply use this number of tick marks. If not, then we increase the height of the scroll bar to accommodate exactly as many ticks as needed, and add a scroll bar.

The location of the labels is determined iteratively beginning with the oldest date. The exact, ideal location is first determined using the dates' timestamps, and is then rounded to the nearest tick mark. If the desired tick mark is already occupied, then the date label is placed on the next available tick mark. If there are no more available tick marks, then the labels are all shifted down by one until space can be made. While this algorithm does not provide an optimal layout, even when space is available for all of the dates, it does provide a quick approximation of the optimal layout without occlusion between the dates. When the page is resized, the layout algorithm is called again for the slider to determine the approximately optimal position of each date label.

The bar chart is computed by iterating over each change within each date to determine how many changes of each type occurred within each date. This information is then used to calculate the height of each bar. In addition, invisible elements containing the ids of each segment changed at that date are added to the page to support the linked highlight and linked navigation features. The width of each bar is calculated such that the date with the most changes will have a bar whose width is equal to the maximum space available. A minimum width of 3 pixels is specified for any bar with non-zero width.

When the highlight method is called for the bars, new elements are drawn on top of the existing bars with no background color, and a single pixel width border. The width of each bar is first determined by counting the number of changes of each type that are in the current highlight selection.

5.2.4 Document Structure View

The icicle plot in the document structure view was made using a very simple open source zoomable icicle [9] made with d3.js. We heavily modified this implementation to convert it to a vertical icicle plot so that it would fit better with the layout of our system, and echo the vertical structure of the law. We modified the existing system which used svg rects to use groups of both rects and text objects, so that we could draw the labels on top. We used jquery.tipsy.js [6] to include the tooltip labels. We also modified the existing implementation to remove the borders around the segments and alternate the luminance of adjacent segments. When the icicle plot is first drawn, we set its height to be the greater of the space available, or the minimum number of pixels needed to guarantee each segment will have one pixel height. Each time page is resized, we recompute this height, and the width available to redraw the icicle plot.

The double click function for the icicle plot required us to edit the default behaviour of events in Javascript. In the standard implementation, when the double click event is dispatched, the click event is also dispatched. This means that when a user double clickes on an element to zoom in, it also causes the main diff view to scroll to that element. A behaviour which seemed to our test users to be an undesirable side effect. The current implementation first waits for 350 milliseconds to see if the user will click a second time before scrolling to the selected segment.

5.2.5 Summary View

The summary view is currently implemented in a brute force manner. This is due both due to time constraints, and because our current knowledge of Javascript, along with some quick Google searches, indicated that there may not be an easy solution. Currently, each time the document structure view is zoomed in or out, or scrolled, a function is called which checks to see if each element in the icicle plot is within the current scroll frame. If it is not, then it is added to the corresponding bar. The maximum number of changes is counted during the bindWithDiff function, which is used to normalize the bars. Future versions could make use of an event that is only triggered when element come in or out of view, however our initial Google searches indicated that this type of event could be simulated by using a similar brute force method to our implementation. Possible future improvements on this implementation could also include a data structure which makes use of the fact that the position of each segment in the icicle plot is highly structured, which could be used to significantly or completely reduce the number of checks that would need to be made to each element's position.

6 Results

The main tasks supported by Visual Laws can be separated by the two different types of potential users of the system: legislators and academics who study law.

6.1 Scenario of Use: Legislator

Fig. 9: The opening screen when VisuaLaws is loaded.

A user who is a legislator will want to be able to see what the law looked like at a specific date as well as compare it with the current version of the law. This will be done by selecting a law to view, see Figure 9. Once the law is loaded, the user will bring both sliders on the timeline slider on the right to the date of interest. The user will then be able to see the version of the law at that date without any markup, as in Figure 10. The user will then move one of the sliders back up

Fig. 10: The Supreme Court Act as of April 1, 2008.

to the current date in order to compare the two dates. Once the years are selected, the document structure view on the left and the main diff view containing the law document will both be updated to encode the sections added, removed and edited using colour highlighting and text strike-outs, as in Figure 11. The user will then click on a segment of interest, in this case Section 11, in the document structure view to navigate to it automatically in the main diff view. The added lines in the segment will be highlighted in green and removed lines will be crossed out in red allowing the user to quickly identify the differences between the two versions of the document, see Figure 12.

6.2 Scenario of Use: Academic

A user who is an academic will first load a law of interest. The user can also easily identify what segments of the law were repealed, added, or changed over the years by looking at the colored segments of the document structure view, see Figure 13. The user may also identify any dates of interest by observing which years have large bars in the scented time-line slider on the right, indicating years of high activity, see Figure 13. The user may hover over the dates to see what segments of the law were affected on this particular date. The hovering action will also allow the user to see if any of the segments affected on that date were also changed in other years see Figure 14. The user can then click on that segment in the document structure view to gather more detail about how it evolved over the years.

7 Discussion

Our tool can successfully support all the tasks outlined above. The five separate views allow the user to see an overview of the evolution of a law as well as drill down to get the specific details of each change. One of the strengths of the system, is that with the exception of the timeline view, the user is presented with a complete summary of the changes selected. That is, when segments are out of view in the document structure view, they are not left out of mind, as they are still summarized by the summary view. Similarly, information outside of the main diff view is summarized both in the document structure view, and the scented scroll bar view.

On the flip side, for laws with more changes than can be displayed at once within the timeline view, there is no summary provided of what is not within the scroll frame, which is a weakness with the current implementation. In addition, when a scroll bar is added to the timeline both date selectors on the timeline may not be visible. Without the date selectors in view, the user may loose track of which dates they are comparing. Despite these limitations, we believe it is still the most effective option, since other possible design choices we considered would have required occluding the dates.

We need to make some significant changes before VisuaLaws can be considered a working tool. First, the number of laws supported currently is minimal. This problem can be fixed by dealing with the issues with the parser. Second, the performance can definitely be improved. The biggest law currently takes about 15 minutes to load. This delay is definitely a factor that needs to be fixed before VisuaLaws is deployed.

This project has taught us many valuable lessons. First, data format is crucial. It is important to know exactly what kind of data you are working with before starting a project. Due to the massive volume of data included in bigger laws, we did not take the time to sift through all the information to find every possible exception or rule that may appear before beginning to write the parser. As a result, the parser usually fails to accurately display all the correct information when a new law is loaded and needs to be slightly altered before use. We also made some false assumptions at the beginning about the data format. For example, we assumed all repealed segments were listed in the current version of the law and simply marked as repealed. We had to make significant alterations to the parser in order to work around these erroneous assumptions. It is also important to know the scope of the data as it will dramatically affect the initial decisions that need to be made in terms of design choices. Though the data set may be large, it is important to take the time to sift through as much of the information at possible to get a general idea of what to expect.

Another crucial less learned is that pixels are a precious resource. We spent the majority of our time building the visual components around a relatively small to medium sized, manually parsed law. It was only after we began visualizing bigger laws that we realized the importance of pixels and had to modify some of our design choices. There are only a fixed number of pixels on a screen and if the number of elements exceeds that, then decisions need to be made. The trade-off is between the visibility of every single item in the data set as opposed to the visibility of the entire system at any point in time. We optimized our use of pixels by alternating colors in the document structure view to differentiate between different consecutive segments. instead of using borders which require valuable pixel real estate.

Colours are hard. There are many aspects to consider when choosing colors such as appropriate hue, saturation, and luminance. Some users may also be colorblind which requires additional consideration when developing a tool that relies heavily upon the use of color as the

Fig. 11: The changes between today, and April 1, 2008 for the Supreme Court Act.

primary identifier. One solution that we employ in our tool is to add redundant visual marks such as underlines and strike-outs. Finally, colours also have the ability to make an interface visually appealing, which confounds the task of finding appropriate colours for a visual encoding.

Future versions of VisuaLaws will address some of the current limitations by improving some of the tools that already exist. Improving the speed of the system for large laws, and supporting the accurate representation of the evolution of all laws on the BC Laws website, are two crucial improvements needed before deployment. These issues can be fixed by improving the parser so that is can handle all data formats. Future versions could be improved by using a new text difference tool designed to identify what portions of text should be marked as added and removed to produce the most salient document. For example if the word “mango” replaces the word “man”, it is unclear whether or not “go” should be considered added, or if “man” should be considered removed, and “mango” added.

Future versions of VisuaLaws may need to more explicitly indicate the differences between enacted, re-enacted, amended, etc. Expert user student would be highly beneficial to determine the effectiveness the tool in its current state.

Future versions will also include some new features that will further support the intended tasks and add to the overall user experience. First, a user will be able to shift-click on various dates on the timeline in order to highlight multiple dates and view the intersection of their corresponding changes. This option will allow the user to easily compare the changes between dates instead of having to alternate between hovering on the the various dates of interest and remembering what changes were highlighted. Additional linked highlighting should be added to the summary view. This highlighting will allow the user know if the change they have selected is currently out of the document structure scroll frame and indicate whether it is above or below the current scroll frame.

Future versions will also derive and display “themes“ common within change dates. For example, in the Marriage Act, on a November 24, 2011, various segments of the law were changed to better support same-sex marriages. The tool will make these themes apparent to the user.

8 Conclusion

VisuaLaws effectively displays all of the changes made to a law over time using five linked views. In the main diff view the user is able to drill down to see the specific text of the law and how it has changed, while a summary view is provided with the scented scroll bar. The timeline view acts both as a summary of changes over time as well as providing user interface controls for the user to select which dates they would like to compare. The document structure view’s dense display allows the user to quickly scan through the structure of the document to build a mental model of the changes between the two versions of the law. Linked navigation in this view allows the user to easily navigate to interesting segments in the main diff view for more details. To help ensure that the user does not forget about changed segments that are not displaying in the current scrollable view of the document structure frame, we use the summary view to show the fraction of changes not displayed. VisuaLaws employs the standard visual encoding techniques of overview and detail views at several levels, with linked navigation and highlighting where each view can become the master controlling the response in the other views. The linked highlighting between each view allows the user to quickly understand the relationship between elements in each view to build a mental model of how the law has changed over time. The linked navigation and overview detail semantics between each view allow the user to quickly drill down to key segments of interest.

Acknowledgments

The authors wish to thank Tamara Munzner for her helpful suggestions for design improvements, and Halldór Þórhallsson, Jacob Chen, Louie Dinh, Robbie Rolin, and Vaden Masrani for their helpful feedback on the various versions of the tool.

References

[1] Bc laws. http://www.bclaws.ca/. Accessed: 2015-12-17.
[2] CanLii. http://www.canlii.org/en/bc/laws. Accessed: 2015-11-08.
[3] Diff Checker. https://www.diffchecker.com/. Accessed: 2015-11-07.
[4] jQuery-ui. https://jqueryui.com/. Accessed: 2015-11-22.
[5] jQuery-ui Labeled Slider. http://bseth99.github.io/projects/jquery-ui/7-jquery-ui-labeled-slider.html. Accessed: 2015-11-22.
[6] jquery.tipsy.js. http://bl.ocks.org/ilyabo/1373263. Accessed: 2015-12-18.
[7] Microsoft Office Word 2013. https://products.office.com/en-ca/word. Accessed: 2015-11-07.
[8] spin.js. http://spin.js.org/#v2.3.2. Accessed: 2015-12-18.
[9] Zoomable icicle. http://bl.ocks.org/mbostock/1005873. Accessed: 2015-12-18.

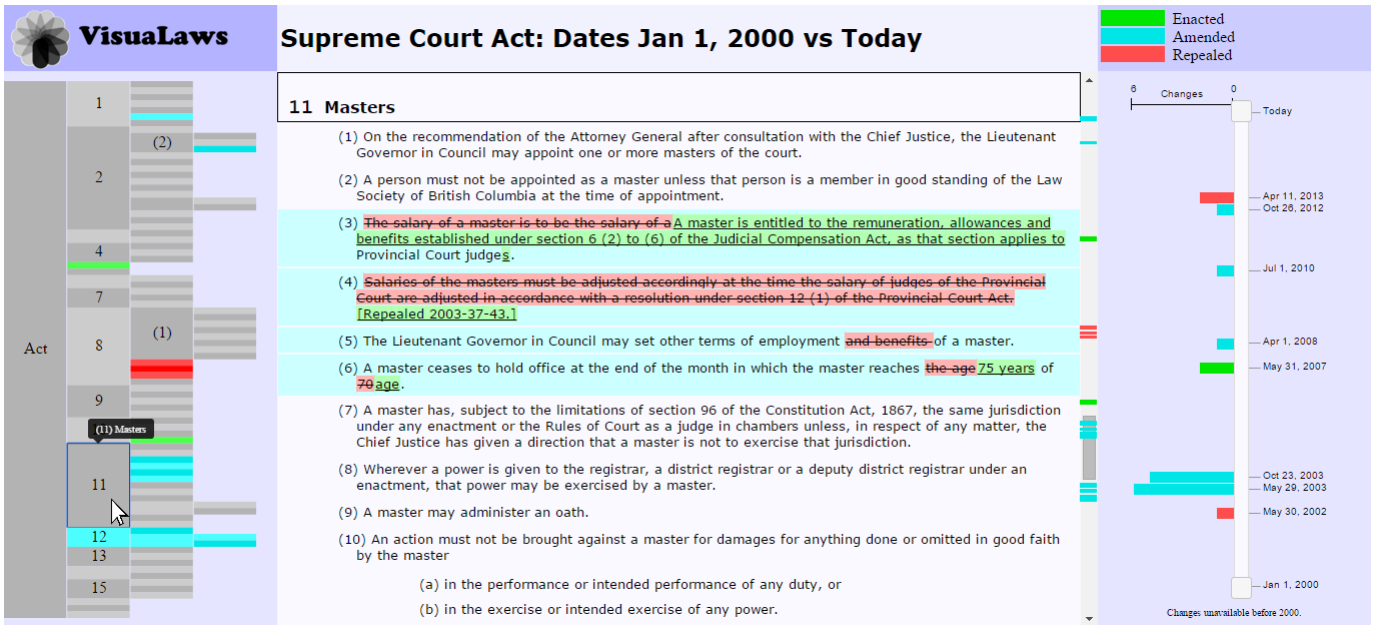


Fig. 12: By clicking on Section 11 in the document structure view the user is able to navigate to it in the main diff view

[10] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[11] K. De Volder. Jquery: A generic code browser with a declarative configuration language. In *Practical Aspects of Declarative Languages*. 2006.

[12] J.-D. Fekete and C. Plaisant. Interactive information visualization of a million items. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*. IEEE Computer Society, 2002.

[13] N. Fraser. Diff.js. https://neil.fraser.name/news/2006/03/19/. Accessed: 2015-11-22.

[14] P. M. P. C. Guerra-Gómez, J.A. and B. Schneiderman. Visualizing changes over time in datasets using dynamic hierarchies. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*.

[15] S. Havre, P. Whitney, and L. Nowell. Themeriver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*.

[16] J. Jones and M. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

[17] T. Munzner. *Visualization Analysis and Design*. A K Peters Visualization Series, CRC Press, 2014.

[18] T. Munzner and F. Guimbreti'ère. Treejuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. *ACM Transactions on Graphics (TOG)*, 2003.

[19] H. Schulz, S. Hadlak, and H. Schumann. The design space of implicit hierarchy visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*.

[20] Y. Tu and H. Shen. Visualizing changes of hierarchical data using treemaps. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*.

[21] F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

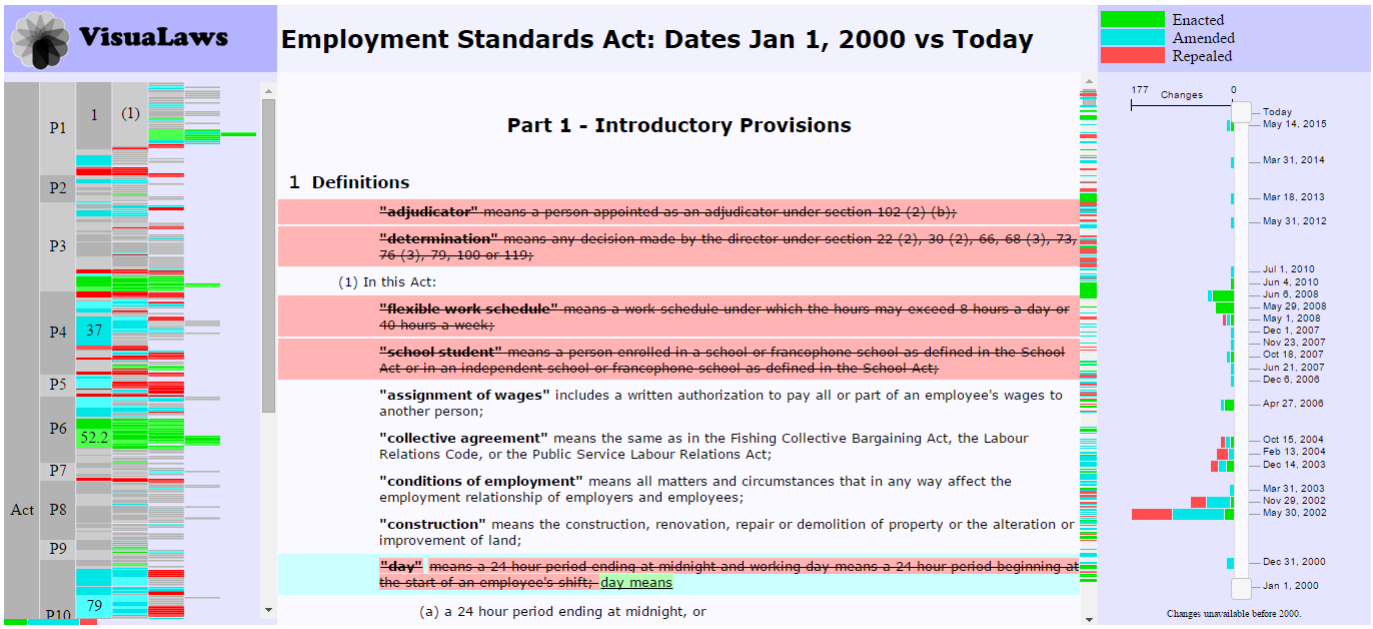


Fig. 13: By clicking on Section 11 in the document structure view the user is able to navigate to it in the main diff view

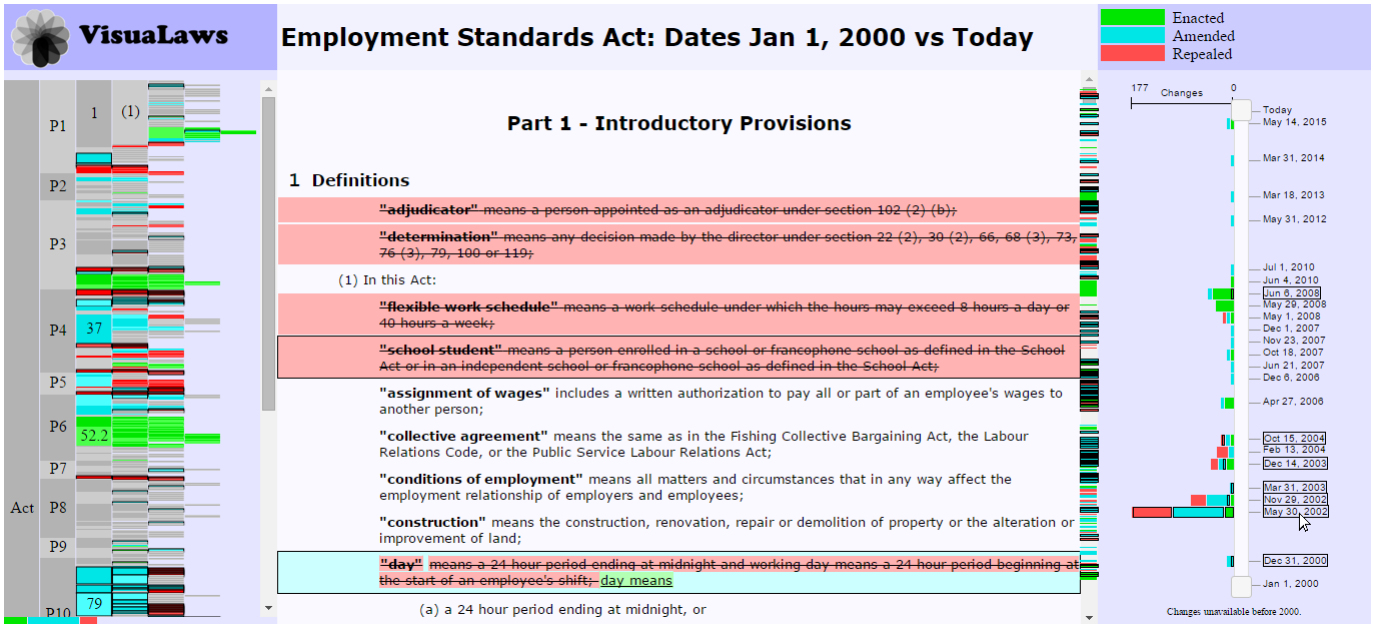


Fig. 14: By hovering over the date May 30, 2002, the user is able to see all of the segments changed on that date, as well as the other dates which have changed any of those segments.