

Institut für Visualisierungsinstitut  
Universität Stuttgart  
Allmandring 19  
D-70569 Stuttgart

Master's thesis Nr. 3340

# **Implementation of an Interactive Visualization Tool for Analyzing Dynamic Hierarchies**

Christine Louka

**Course of Study:** Infotech  
**Examiner:** Prof. Dr. Daniel Weiskopf  
**Supervisor:** Dr. rer. nat. Michael Burch

**Commenced:** May 15, 2012  
**Completed:** November 14, 2012

**CR-Classification:** H.5.2



# Abstract

Many real world examples can be found that deal with hierarchical data. Software systems typically consist of packages, directories, subdirectories, files, classes, and functions. Phylogenetic trees structure biological species into a hierarchical organization. Visualizing such static hierarchical data has been in focus of Information Visualization for many years. Visually encoding and understanding of evolving hierarchies still remains a challenging task. Since hierarchies may grow huge and may evolve over a long time producing many time steps, we make use of a side-by-side and aligned representation of Indented Pixel Tree Plots. To achieve a mental map preserving overview-based diagram we show the dynamics of a hierarchy by a static representation and illustrate the changes between subsequent hierarchies by special links. Interactive features make the data manipulable and navigable in all dimensions.



# Acknowledgement

First and foremost I would like to thank God for his blessings, support and the strength he gave me.

It gives me great pleasure in acknowledging the support, help and guidance of Dr. rer. nat. Michael Burch. Thank you for your fast e-mail replies even when you were on business trips or holidays.

I also want to thank my friends in Stuttgart whom I cannot find words to express my gratitude to, Mirna Aiman, Michael Guirguis, Mariam Hassib, Ghada dessouky, Youssef Ghaly, Pierre Ibrahim, Mina Metias, Ahmed Halawa and Amr yassin. It was a two year journey and they have always been there supporting and helping me all the way, I cannot thank them enough.

This thesis would not have been possible without my parents' encouragement and insistence on travelling and doing my masters abroad, so a big thanks to my mother and my father as well as to my twin sister Sandra Louka who was there whenever I needed her the most with her love and patience.

And last but not least, I would like to thank my friends and family in Egypt for their prayers and support from long distance. They always cared on asking me how I am doing and always encouraged me.

Each of you can share in this accomplishment, for without your support it would not have been possible.



# Contents

<b>List Of Figures</b>	<b>VIII</b>
<b>List Of Abbreviations</b>	<b>IX</b>
<b>1 Introduction</b>	<b>1</b>
1.1 A Motivating Example . . . . .	1
1.2 Aim of the Project . . . . .	3
1.3 Remainder Of The Master Thesis . . . . .	3
<b>2 State Of The Art</b>	<b>5</b>
2.1 Hierarchy Visualization Systems . . . . .	5
2.1.1 Node-link diagrams . . . . .	6
2.1.2 Treemaps . . . . .	7
2.1.3 Layered icicle . . . . .	11
2.1.4 Indented layout . . . . .	17
2.1.5 Hybrid Representation . . . . .	19
2.2 Technique for changing hierarchical data . . . . .	20
2.3 Time-Series Visualization . . . . .	20
2.3.1 Animation vs. Static . . . . .	20
2.3.2 Mental Map . . . . .	21
<b>3 Case Studies</b>	<b>23</b>
<b>4 Project Architecture</b>	<b>33</b>
4.1 Process Overview . . . . .	33
4.2 Class Diagram . . . . .	34
4.3 Features and Functionalities . . . . .	37
<b>5 Implementation</b>	<b>39</b>
5.1 Newick File Parser . . . . .	39
5.1.1 Node . . . . .	42
5.2 Search Engine . . . . .	42
5.3 Collapse/Expand Algorithm . . . . .	43
5.4 File Hierarchies Comparison . . . . .	44
5.4.1 Comparison Algorithm . . . . .	44
5.5 Visualization . . . . .	49
5.6 IPTPtool . . . . .	54

<b>6</b>	<b>Conclusion</b>	<b>57</b>
<b>A</b>	<b>User Manual</b>	<b>59</b>
A.1	Menu Bar . . . . .	59
A.2	Tool Bar . . . . .	61
A.3	Configuration Panel . . . . .	63
A.4	Detail Panel . . . . .	65
A.5	Control Panel . . . . .	69
A.6	Hierarchy Panel . . . . .	70
A.7	Bar Chart Panel . . . . .	71



# List of Figures

1.1	Node-link diagram of a hierarchy in a top-down layout . . . . .	1
1.2	The changes of a hierarchical organization . . . . .	2
2.1	Visual metaphors for hierarchical data . . . . .	5
2.2	Node-link diagram . . . . .	7
2.3	Different Treemap layouts . . . . .	8
2.4	Nested Treemap . . . . .	9
2.5	Cushion Treemap . . . . .	10
2.6	Pebble Treemap . . . . .	10
2.7	Voronoi Treemap . . . . .	11
2.8	Cartesian layout . . . . .	12
2.9	Sunburst layout . . . . .	13
2.10	Information slices . . . . .	14
2.11	Angular detail technique . . . . .	15
2.12	Operations implemented for interactivity on hierarchical structures . . . . .	16
2.13	Windows Explorer . . . . .	17
2.14	Indented pixel tree plot . . . . .	18
2.15	Elastic hierarchy . . . . .	20
2.16	IPTPs compared . . . . .	22
3.1	IPTP representing the 'NCBI taxonomy' highlighting a subregion . . . . .	24
3.2	IPTP representing the 'NCBI taxonomy' illustrating the deepest leaf nodes . . . . .	25
3.3	IPTPs of the 'dblp.newick.100' file . . . . .	26
3.4	Hierarchy Legend . . . . .	26
3.5	IPTPs of the 'dblp.newick.100' file after comparison . . . . .	27
3.6	IPTPs of the 'dblp.newick.100' file after applying filters . . . . .	28
3.7	IPTPs of the 'dblp.newick.100' file after a node search . . . . .	29
3.8	IPTPs of the 'dblp.newick.100' file after applying the time filter . . . . .	29
3.9	IPTPs of the 'dblp.newick.100' file after multiple filter application . . . . .	30
3.10	Information on specific IPTPs . . . . .	31
4.1	Process overview diagram . . . . .	33
4.2	Class diagram (1) . . . . .	35
4.3	Class diagram (2) . . . . .	36
5.1	Tree representation of a parsed newick file . . . . .	39
5.2	Collapse/Expand illustration . . . . .	43
5.3	Delta . . . . .	46

5.4	The Graphical User Interface of the Tool . . . . .	54
A.1	The Graphical User Interface of the tool . . . . .	60
A.2	Tool bar . . . . .	62
A.3	Zoom slider . . . . .	64
A.4	Filter . . . . .	65
A.5	Search engine . . . . .	66
A.6	Resolution mode . . . . .	66
A.7	Hovered nodes details area . . . . .	67
A.8	General file details . . . . .	68
A.9	Delta . . . . .	68
A.10	Graph legend . . . . .	69
A.11	Control modes . . . . .	70
A.12	IPTPs display area . . . . .	70
A.13	Bar chart . . . . .	71

# List of Abbreviations

<b>NLD</b>	Node-Link Diagram
<b>IPTP</b>	Indented Pixel Tree Plot
<b>RSF</b>	Radial space-filling
<b>GUI</b>	Graphical User Interface
<b>DBLP</b>	DataBase systems and Logic Programming
<b>NCBI</b>	National Center for Biotechnology Information
<b>JPEG</b>	Joint Photographic Experts Group



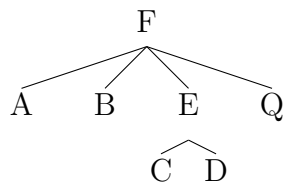
# Chapter 1

## Introduction

Hierarchical data occurs in many application domains. File systems consist of directories, subdirectories and files. Also in the software development process, a hierarchically organized project is needed to better maintain the whole system and prevent bugs. In the field of biology, the NCBI taxonomy expresses which organisms belong to which sub-hierarchies and build a phylogenetic tree. The visualization of static hierarchies has been in focus of research for a long time [17] but evolving hierarchical data is still a big challenge for the visualization community.

### 1.1 A Motivating Example

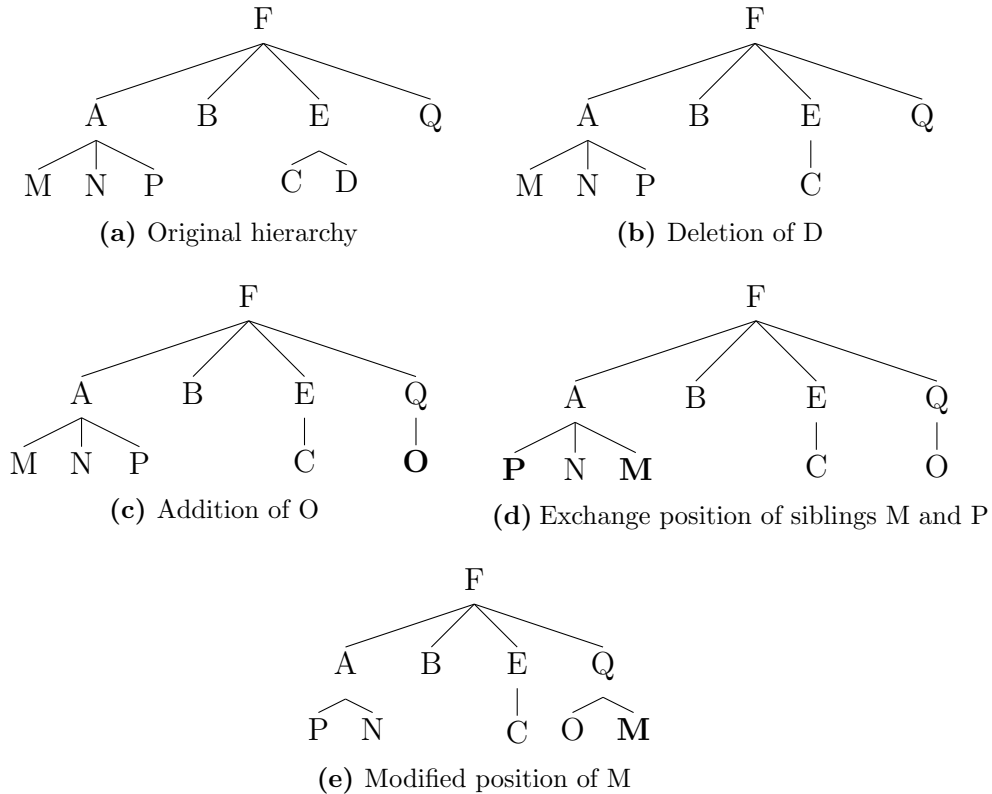
In the domain of software development and programming we have to deal with large amounts of data. Typically, software is composed of entities that are hierarchically organized into code blocks, methods/functions, classes, files, subdirectories, directories, and packages. Figure 1.1 shows an example of a hierarchy containing 7 vertices at a depth of 2. The node labelled with "F" is the root node, "E" is the only inner node and "A", "B", "C", "D", and "Q" are the leaf nodes of this hierarchy.



**Figure 1.1:** Node-link diagram of a hierarchy in a top-down layout

Software systems are not static but they are evolving over time. Consequently, the hierarchical organization is also not static but is changing over time more or less frequently. For example, there maybe added or removed software entities as well as those

that are just moved to another part of the hierarchy which is a typical phenomenon when restructuring or refactoring the software system. An illustration of such changes is shown in Figure 1.2.



**Figure 1.2:** The changes of a hierarchical organization

Telea and Auber [31] have already introduced a system that deals with software systems that illustrate the changes discussed above. Their work focuses on changes of source code overtime whereas the proposed visualization tool is able to work with any kind of evolving hierarchical data.

Hierarchical data can become large in many dimensions depending on the application domain. In software development for example we may have to deal with several million elements on the source code level and several thousand time steps depending on the granularity of the analyzed time interval.

An exploration of the raw textual data is hence a very time-consuming process if not impossible at all. For this reason, visualization is applied to such time-varying hierarchical datasets with the goal to find as many insights in the data as efficient as possible by exploring the perceptual abilities of the human visual system and the strengths for pattern recognition.

## 1.2 Aim of the Project

Hierarchical datasets can be found in many application domains where a large number of items (files, products, employees, stocks, etc.) can be handled and managed more efficiently when they are grouped into larger entities.

Nowadays, huge amounts of data are stored in databases, archives and even clouds, so this makes it important for the person in need of a specific information extracted from a huge pile of data to have an easy user friendly tool that helps visualizing certain data depending on user specifications and needs. Hospitals for example, have large amounts of files storing patients' medical history along with their personal information. Being able to keep track of all these information and keep track of changes is a tiring and challenging process. Changes mentioned could be like having data moved, deleted or added which affects the hierarchy of files and folders. Hence the importance of hierarchy comparison is required.

Small hierarchical structures are very effective to locate information, but the content and organization of large structures is much harder to grasp. So in such cases the visualization of large hierarchical datasets is an important topic in the visualization community.

Comparison of hierarchical information is of importance within various areas; for example, evolutionary biology, human resources and personnel, software development and hospitals as mentioned above [35]. Since human vision is sensitive to patterns, variation of colors and shapes we can have answers to specific questions very quickly. So visualization allows us to focus only on information that is important such as trends and outliers.

The aim of this project is to implement a tool that is capable of visualizing and interacting with large amounts of data in a user friendly manner to have a better and faster understanding of the data. To this end there are many visualization techniques for visualizing and analyzing large datasets. In this thesis we apply the approach of Indented Pixel Tree Plots (IPTPs) as the technique for analyzing dynamic hierarchies since it provides a scalable variant for visualizing hierarchies and allows good comparisons between time steps by aligning the plots side-by-side.

## 1.3 Remainder Of The Master Thesis

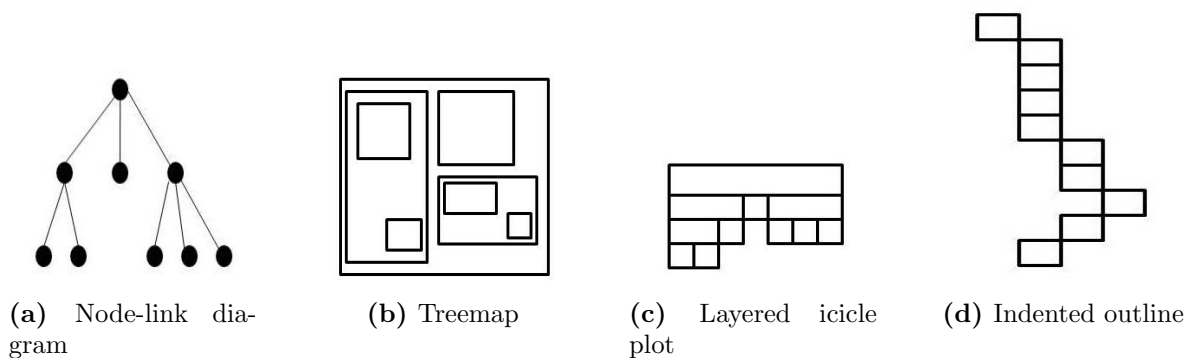
This thesis consists of the following chapters. **Chapter 2** discusses the State of the art in visualizing hierarchical datasets, explaining the different techniques for exploring

large amount of information as well as the different methods of dynamic visualization and how to preserve the mental map. Followed by **Chapter 3** describing the datasets used for user testing and the insight gained after visualizing them. In **Chapter 4**, the project architecture is described embracing the process overview, the class diagram and the features and functionalities introduced by the tool. **Chapter 5** discusses the implementation of all algorithms and methods and describes the Graphical User Interface (GUI) of the tool. Finally, **Chapter 6** contains the conclusion, the limitations and provides possible directions for future work.



# Chapter 2

## State Of The Art



**Figure 2.1:** Visual metaphors for representing hierarchical data: (a)Node-link diagram. (b)Treemap. (c) Layered icicle plot. (d) Indented outline.

There is more than one method of hierarchy visualization available nowadays so it would be more effective to have them organized into categories. The first category contains node-link diagrams, the second nested enclosure techniques (space-filling technique [6]) like the Treemap, the third category the stacking approaches (Layered icicles), leaving the indented outline style in the fourth category, see Figure 2.1(a)-(d). Even though each category has its own advantages and disadvantages the only identification of perfection of such a category can be investigated by conducting comparative user studies for different user tasks that needs to get accomplished using the different categories mentioned.

### 2.1 Hierarchy Visualization Systems

Since the hierarchical data structure has been widely used specially in the Information Visualization area [19], the most common structure would be the tree-like (node-link) structure and it can be used in the visualization of file system, organizational charts,

web sites, decision trees, genealogy and digital libraries. Even though such a structure is widely used and various layout algorithms have been developed as well the main problem there is the efficiency of conveying the topological structure of large trees when visualized.

The main purpose of the visualization techniques proposed is to solve problems associated with hierarchy visualization techniques, i.e., such problems could be as follows:

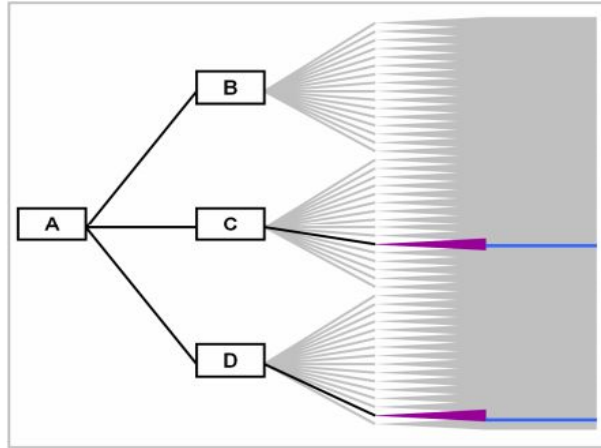
- Using efficiently the available display space while conveying the complete hierarchical structure
- Allowing the user to examine details of various regions of the hierarchy simultaneously
- Enabling the user to easily interact with the hierarchy and perform tasks such as modifying the hierarchy and selecting nodes on which to perform operations

Despite the various advantages of those techniques, however, neither one of them can solve all the problems whereas most recent hierarchy visualization research focused on the challenge of displaying large hierarchies in an easily comprehended form.

### 2.1.1 Node-link diagrams

The node-link diagram (NLD) represented in Figure 2.1(a), is probably the most natural way to display nesting structures [23]. It is the most familiar diagram to users and it is appropriate for displaying the shape and structure of a tree. However, it fails to scale for large datasets. Node-link diagrams are suited better for showing the different levels and depths of a hierarchical structure. For example, Battista et al. [5], and Reingold and Tilford [23] use conventional node-link diagrams to depict relationships between hierarchically ordered elements. The node-link visual metaphor is considered the most widely used, well-established, de-facto standard for hierarchy visualization [2].

Node-link diagrams convey a clear and unambiguous structure of nodes. The parent-child relationship is represented using links and elements which are presented as nodes. However, node-link diagrams distribute nodes unevenly, leaving upper level nodes separated by white space, and lower nodes densely packed. While node-link diagrams show nesting structures very clearly, they consume screen space inefficiently, and do not scale well for large datasets. This means they are only beneficial when small trees need to get displayed, as a result many approaches have been proposed to supplement the node-link diagrams. The well-known alternatives include Treemaps, cone trees, and the hyperbolic browser [38].



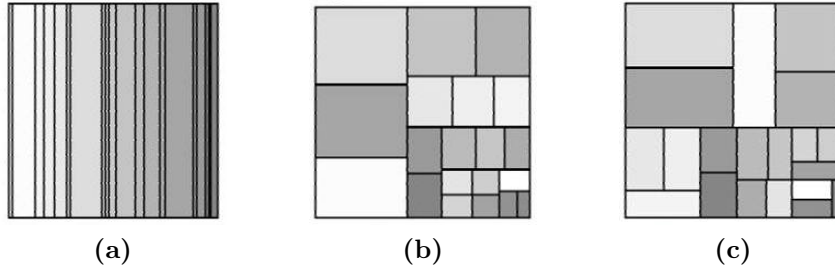
**Figure 2.2:** Node-link diagram [38]

### 2.1.2 Treemaps

Treemaps are enclosure diagrams belonging to the space-filling techniques. Instead of using adjacency or node-link for hierarchy representation they use the concept of containment as a visual metaphor [12].

#### Variants of Treemaps

The most common way to visualize hierarchies is by using trees, where edges describe the parent-child relationship between nodes [16]. However, Treemaps utilize a more space-constrained technique to visualize hierarchies, which solves the problem of normal trees having lots of non-utilized display area [3]. This is by displaying nested sequences of rectangles, whose areas represent the attributes of the dataset, hence allowing an easier comparison of node sizes. This implies that Treemaps fall under the nested containment category (Figure 2.1(b)). This category differs from the three other categories where child nodes are drawn inside parent nodes, however they are represented outside in the other forms, whereas shades of colors represent the depth of each attribute. Treemaps are space-efficient [25] and perform well at giving an overview of very large trees, visualizing thousands of nodes constructing a hierarchical dataset [6]. However, and unlike node-link methods, it does not allow a clear node structure representation, especially when visualizing a balanced tree where all parents have the same number of children and leaves have same node sizes as this results in a treemap having the form of a regular grid. This unclear structure makes the different levels of the tree harder to perceive and distinguish. Only leaf nodes are clearly perceived since they overlap the parents' areas. This drawback makes it less familiar to users than the node-link technique since it could be more difficult to interpret a hierarchy unambiguously.



**Figure 2.3:** Different Treemap layouts: (a)Slice-and-dice. (b)Squarified. (c)Pivot by split size [28].

There are several algorithms to create treemaps [34, 28, 25, 7, 33, 32]. The original treemap layout is called slice-and-dice [Shneiderman 1992], illustrated in Figure 2.3(a). It employs parallel long lines to divide a rectangle representing an item into smaller thin elongated rectangles representing its children [25]. The orientation of the line is switched from horizontal to vertical and vice versa, whenever a new level is introduced. Despite the algorithm’s simplicity, the long skinny rectangle lines with a high aspect ratio between width and height could be sometimes harder to recognize, select and compare in size and label.

Treemaps have been widely used in many domains starting from financial analysis to sports reporting [25] like visualization of a tennis match [27] as well as in mainstream media, such as in displaying news headlines. However, an unexpected advantage of treemaps is the ease at which shallow nodes can be seen no matter how deep the subtree under a node may be, since treemaps tend to allocate more screen space to shallow nodes.

### Squarified Treemaps

As an alternative, a ”squarified” treemap algorithm [7], illustrated in Figure 2.3(b) introduced by Jarke van Wijk, and a ”cluster” treemap method in Figure 2.3(c), described in Wattenberg having simple recursive algorithms, were introduced to reduce the overall aspect ratios to have the long skinny rectangles refined to square-shaped with aspect ratios close to one for better node detection and comparison [25]. Although these methods presented a clear refinement from the aspect ratio point of view, they sometimes lack the clear hierarchy structure preserved in the original slice-and-dice layout. Other known drawbacks [28] reported are that changes in the dataset can cause dramatic discontinuous changes in the layouts produced by both cluster and squarified treemaps. The second drawback of these layouts is that they cannot retain the given order of data, while many datasets contain ordering information that is helpful for seeing and distinguishing certain

patterns or for locating particular objects in the layout.

## Ordered Treemaps

Ordered treemaps [28] were then introduced to address the drawbacks of the cluster and squarified methods presented previously. The ordered treemaps algorithm ensures that nodes near each other will in fact, be placed close to each other in the layout. Hence, they preserve the order of nodes displayed while keeping the aspect ratio as low as possible. Moreover, they provide a smooth change in the treemap layout when data is dynamically changing over time (data is getting updated), so it is easier for long-term users to recognize nodes being displayed and to easily allocate them.

## Nested Treemaps

A small remedy developed to overcome the challenges of the treemap layout were the nested treemaps [16]. Instead of having the rectangle subdivided into smaller rectangles, a new rectangle is drawn inside the parent rectangle, and in turn, is subdivided into smaller rectangles representing the child nodes. This results in each group of siblings being enclosed in a margin which facilitates the recognition of parent nodes and layout structure. However, when the hierarchy is getting deeper, it requires more effort to view it easily.



Figure 2.4: Nested Treemap [12]

## Cushion Treemaps

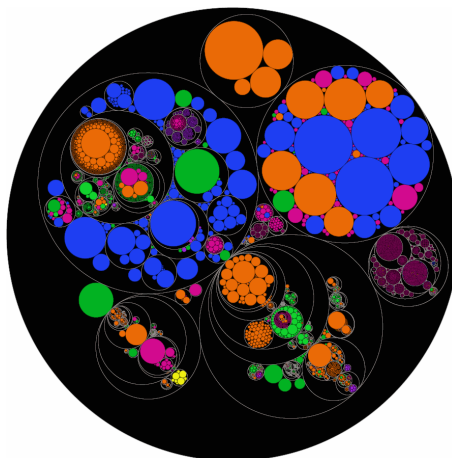
In order to provide the users with a better representation of the visualized structure of the treemap, cushion treemaps [33] were brought out. This type of treemaps uses shading

to provide an easier interpretation of the hierarchical structure, since human perception of shades variation is shown to be very fast [15].



**Figure 2.5:** Cushion Treemap [33]

## Pebble Treemaps

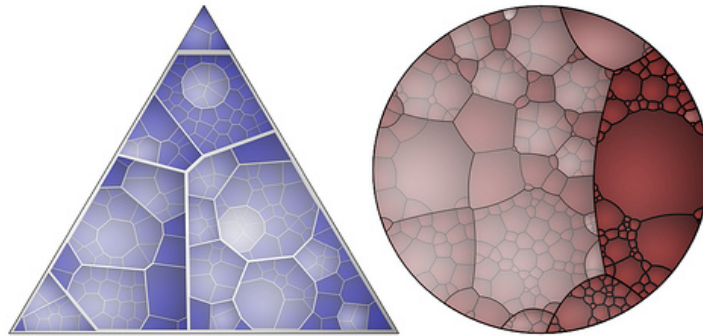


**Figure 2.6:** Pebble Treemap [36]

”Pebble Treemaps” also known as ”Circular Treemaps” or ”Radial space-filling” technique, is a nested circles representation introduced by Wetzel [36], as a refinement to the rectangular treemaps. This approach can better reveal the hierarchy structure and achieve an aspect ratio close to one. However, this approach does not utilize space as efficiently as the rectangular-shaped treemaps and a circle size does not reflect the element (file) but it represents its size.

Colorizing of circles can differ depending on certain attributes. For instance, Wetzel represented a directory in which different files and circles were coloured depending on the file type (images, documents, etc.) as shown in Figure 2.6.

## Voronoi Treemaps



**Figure 2.7:** Voronoi Treemap [3]

All treemaps described above are restricted to axis-aligned rectangles in their representation. Balzer et al. [3] developed a variation of the treemap algorithm which utilizes arbitrary polygons such as triangles and circles as shown in Figure 2.7 instead of rectangular shapes. Its advantages are that the aspect ratio between width and height is closer to one where treemaps lack such advantage, another upside of that structure is that there are no overlappings of nodes and boundaries between hierarchy levels and can be notably observed leading the structure to be better identified <sup>1</sup>.

The nodes in the hierarchy levels are represented by a set of polygons which forms a treemap layout at the end. Polygons support the usage of Voronoi tessellations <sup>2</sup> for their subdivisions [4].

### 2.1.3 Layered icicle

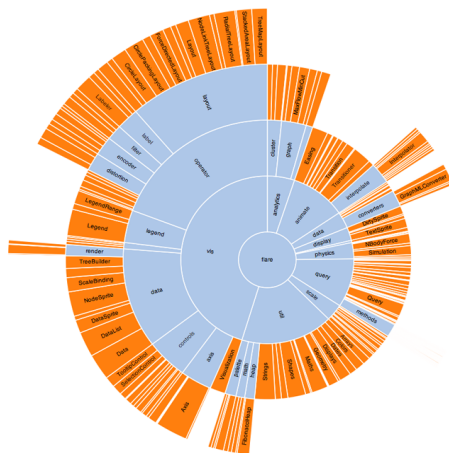
A layered icicle diagram proposed by Kruskal and Landwehr [18], uses a space-filling visualization like a treemap. There are two styles reported in literature that represent this diagram: a 'Cartesian style' and a 'Radial style'. A description for both styles is provided next.





## Radial style

This is another type of "Icicle Layout", but with a circular representation. This representation is also called "Sunburst Layout". In this layout, attributes constituting the hierarchy are arranged in a radial form, where the root element positioned on top of the hierarchy is placed at the center and deeper levels are placed farther away from the center. The size of an element is represented by the angle it subtends, hence, the bigger its size the wider the angle. Color for each element can be given to represent the element type if we are representing a file directory for instance [29]. An example of this layout is shown in Figure 2.9.



**Figure 2.9:** Sunburst layout [12]

Radial space-filling (RSF) diagrams [13, 5] compromise between space-efficiency utilization while maintaining the topological tree structure visibility. Evaluations held by Stasko and Zhang [29] using their RSF tool, "Sunburst", proved the ability of RSF in conveying the tree structure over treemaps. However, an RSF technique fails to capture the details for nodes with high depth level values, and when the hierarchy is large, the small slices are hard to determine. Focus+context techniques are used to overcome these drawbacks as will be described shortly.

Three methods were introduced to overcome these limitations and explore small parts of the hierarchy displayed. Andrew's and Heidegger's two semi-circular approaches [1], Stasko and Zhang's angular detail, detail outside, and detail inside approaches [30], which presents an enhancement of Andrew's approach, aim to overcome the discussed limitations. At last, InterRing that was presented by Yang et al. [37] in 2003 to tackle the drawbacks of both mechanisms is another method. An explanation of all three methods will follow next.

## Information slices

The Information slices approach is a visualization technique introduced by Andrews et al. [1] to visualize and manipulate large hierarchical data. Information is represented using semi-circular discs. Different discs represent multiple levels where higher depth levels are always placed at the periphery of the semi-circle. Figure 2.10 shows a prototype of information slices they implemented for visualizing the hierarchical tree structure of a file system.

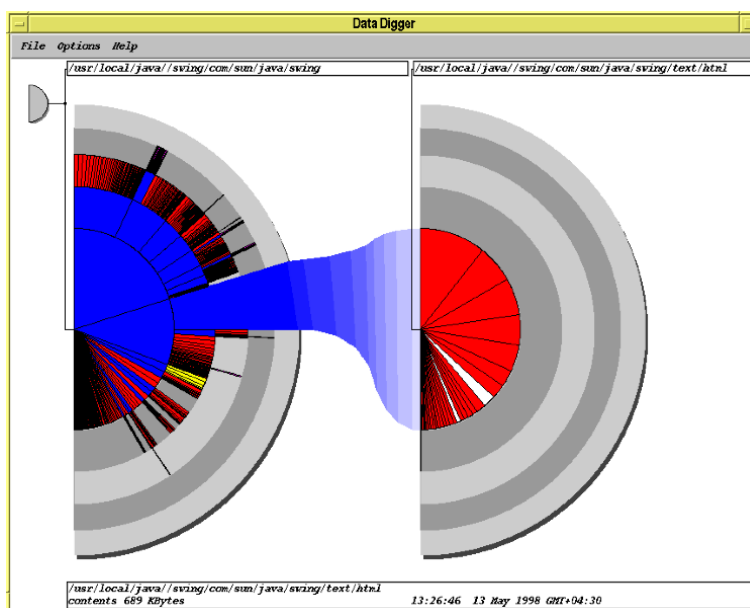


Figure 2.10: Information slices [1]

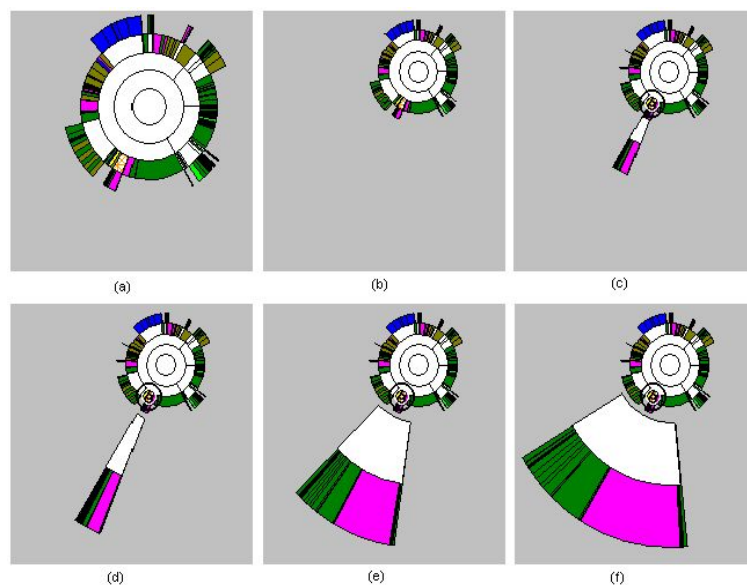
This approach allows the user to expand a certain area on the displayed hierarchy. Two discs can only be shown at once; further expanding removes the leftmost disc from the main panel and is only shown as an icon and the new expansion is observed at the right.

The user is also able to set some configurations such as how many levels to display on each disc, or to display children in which order, i.e., alphabetical or by size.

## Focus+Context

Three methods were proposed to have a smoother and more flexible alternation between global and detailed hierarchy display than Andrew's and Heidegger's semi-circular approach. The techniques are similar in that by clicking on an item, it is focused and observed in the same display of the entire hierarchy. Nevertheless, they differ in how they display the focused area. Each has its own advantages and drawbacks. The three methods are as follows [30]:

- **Angular detail method:** The entire hierarchy shrinks and is moved to the top right corner of the display screen, and the item selected is focused and placed at the center of the display. This technique requires more space, although it looks natural to the user, see Figure 2.11 for clarification.
- **Detail outside method:** The overview shrinks at the center of the display while the selected item is enlarged and placed in form of a new circular ring around the overview.
- **Detail inside method:** The overview is shrunk and widens to have the focused area placed inside it, i.e., at the center of the overview.

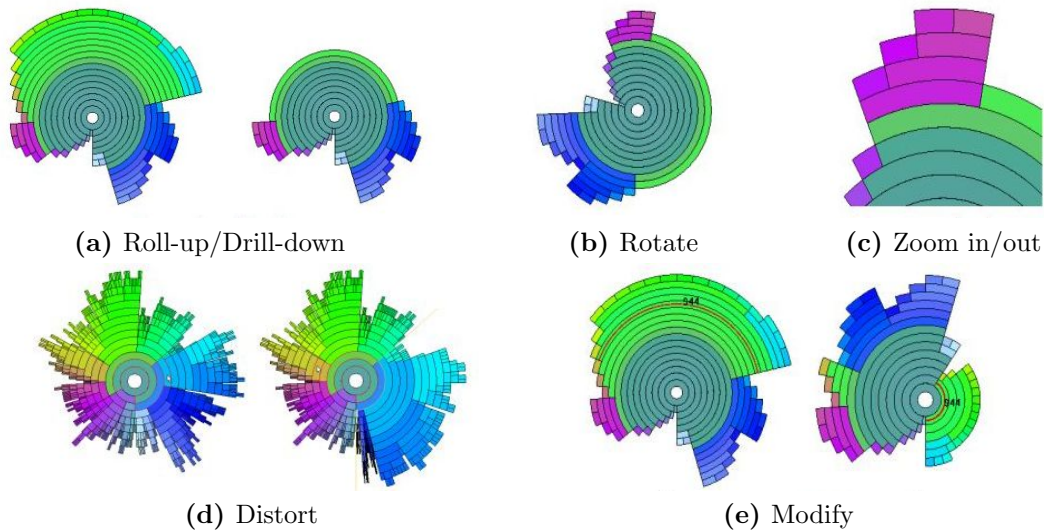


**Figure 2.11:** Sequence of frames from the Angular detail technique, allowing a viewer to focus on small peripheral files [30].

Despite the better observation of details these techniques offer, they suffer from some drawbacks [37]. They cannot preserve the user’s mental map (explained in the following chapter) because of the big visual shift on the original overview observed after an item is focused. Complex animation is needed in order to follow the changes operated on the original overview. They also do not use space as efficiently as the original RSF, and they cannot handle multiple foci.

### InterRing

Previously described RSF techniques do not offer as many interactivity as offered by tree nodes and text-based hierarchy visualization systems. Yang et al. [37] proposed a new



**Figure 2.12:** Operations implemented for interactivity on hierarchical structures [37]

RSF technique "InterRing" to enable users to visualize, modify, and perform selection on the hierarchy and to overcome the drawbacks of Stasko's and Zhang's three mechanisms discussed previously using a new distortion approach.

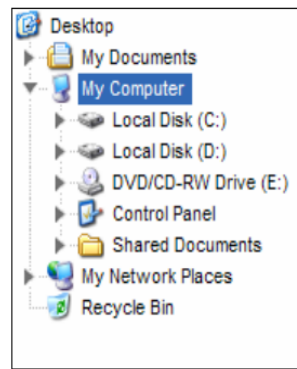
Operations implemented by InterRing for user interactions are as follows:

- **Selection:** The process of selecting multiple nodes for further processing, allows the user to isolate a set of nodes in the hierarchy that can then be highlighted, masked, moved, or deleted.
- **Reconfiguration:** The ability to adjust the hierarchical structure means that users can move a subcluster from one cluster and place it in another to improve the quality of the hierarchy.
- **Drill-down/Roll-up:** The process of exposing/hiding subhierarchies helps users to only show objects of interest and prevent other objects from showing.
- **Pan, zoom, and rotation:** The process of focus, scale, and orientation adjustment to the hierarchy currently on display allows panning and zooming, meaning that users can enlarge the context displayed and examine the details of the hierarchy. Rotating allows users to rotate clusters of interest to specific angles and avoids cluttering the labels of the selected clusters.
- **Distortion:** The process of enlarging certain parts of the hierarchy without affecting the context of the total display. It allows users to have multiple foci, provides

an easy way to follow changes and does not need extra space for the focus+context display.

#### 2.1.4 Indented layout

Indented tree layouts are used excessively by operating systems to represent file directories. They place all items along vertically spaced rows and use indentation to represent the parent-child relationships. Windows Explorer is a classical example of a tree outline structure as shown in Figure 2.13.



**Figure 2.13:** Windows Explorer [22]

An indented outline structure also allows efficient interactive exploration of the tree to find specific nodes. Despite the big vertical space needed for visualizing the hierarchy, the positioning of nodes separately on horizontal lines makes it beneficial to add information to the right of each corresponding node [12].

Indented Pixel Tree Plots (IPTPs), a new hierarchical tree visualization introduced by Burch et al. [8, 9] is a technique that is based on the visual metaphor of indented outlines present in graphical file browsers and everyday software developers' source code. The fact that indented outline approaches are popular in file browsers such as Microsoft Explorer makes it familiar to the user to understand and easily read the hierarchy representing the data [8].

The IPTPs are in a way similar to reading a text fragment where it must be read from left to right to understand the hierarchical semantics, hence they are labeled as a one-and-a-half dimensional visualization approach [8].

## IPTP representation

IPTPs represent inner vertices using vertical lines and employ horizontal lines to represent leaf vertices. Each horizontal line represents a different level. Each node in the hierarchy can be mapped to a certain horizontal line according to its tree level. Edges are represented only implicitly by the vertically and horizontally aligned structure of the plot. Adding edges would be considered superfluous as Tufte mentioned - unneeded information that by discarding, would not affect the understandability and readability of the graph. Parent-child relationships are expressed by indentation of the corresponding geometric shapes with respect to the hierarchical levels of the respective parent and child vertices.



**Figure 2.14:** Indented pixel tree plot [9]

A user study was conducted by Burch et al. [8] to investigate the readability of an IPTP in comparison with a node-link diagram (NLD), - also known as the de-facto standard for hierarchy visualization - in a static way, without colour gradient and any interactive features. The experiment was performed in a laboratory that isolates any distractions, and had been applied on 30 participants. These participants were a mixture of males and females of computer science and engineering backgrounds. Some of them had some background on the concept of visualization techniques and some had not heard of it yet. All participants were tested to ensure they have normal colour vision. Each participant was assigned three tasks. The tasks assigned are described as follows:

- **T1:** Finding the least common ancestor of two leaf vertices
- **T2:** Checking if elsewhere in a plot there exists an identical subhierarchy
- **T3:** Estimating the larger subhierarchy

Prior to the experiment, a 10 minutes training was done to the participants to assure their understanding on both techniques, IPTPs and NLDs. Then, an average of 15 minutes was given to each participant in the evaluation and each of them had to perform all 3 tasks using both techniques (IPTPs and NLDs) with seven trials with different datasets (different tree sizes). After finalizing the tests, participants were given the opportunity to identify their preferred technique by filling in a questionnaire.

The results of the study conducted were analyzed over all dataset sizes applied on all three tasks. The analysis was performed from different perspectives. Completion time, accuracy and the overall preference for each participant, were the perspectives taken. Concerning the completion times, it was found that in tasks T1 and T2, the average time taken to complete both techniques was similar, no significant difference. The significance differed in T3, where it was found that for small and large datasets NLDs were faster to read while for medium sized datasets, IPTPs had very high significance. The average calculated indicated that neither of both techniques was significant in that concern. Concerning the accuracy, statistics shows that no significance was found. Regarding participant's overall preference, it was found that for small datasets, participants preferred the node-link diagram and for large datasets IPTP was found to be more useful.

Andrews et al. [2] also conducted a user study to compare four hierarchy browsers which are the Windows Explorer style tree view, the information pyramids, the treemap and the hyperbolic browser. Task analysis was performed which involved 32 test users where each user performed eight tasks on each browser. Task completion time, subjective ratings and overall preference data were collected. Despite having no significant differences in performance, users significantly preferred the tree view browser.

### 2.1.5 Hybrid Representation

There are also some visualization techniques that use a combination of node-link and treemaps methods like "elastic hierarchies" [38] and "space-optimized tree visualization" [21]. These techniques offer a trade-off between the space-efficiency that characterizes the treemaps and the clear structure display offered by the node-link diagram.

#### Elastic hierarchy

An elastic hierarchy allows users to examine the content of a treemap in more detail and select nodes within it more easily. It also allows the user to change the representation of the hierarchy at any time. Accordingly, this hybrid representation has the potential to flexibly combine the familiarity and clarity of node-link diagrams with the space savings of treemaps. Selection within a treemap is usually difficult because internal nodes are covered by their descendants. Elastic hierarchies solve that problem by implementing a selection technique by showing several tabs to select from corresponding to different levels in the treemap. Each tab causes its level to highlight and allows the users to examine the nodes at that level.

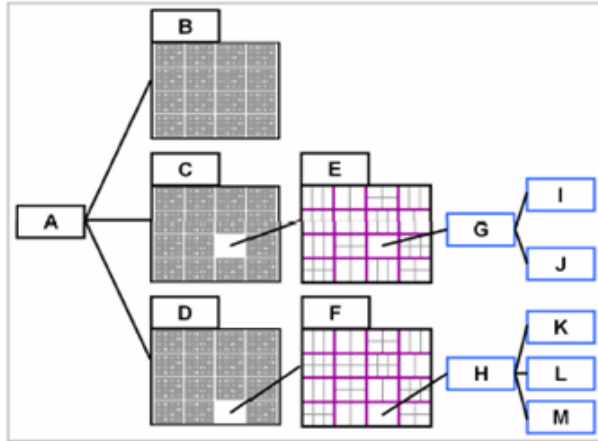


Figure 2.15: Elastic hierarchy [38]

## 2.2 Technique for changing hierarchical data

Code Flows [31], a visualization technique for analyzing source code structure evolution, uses a vertical icicle plot where nodes are ordered by the order of the code lines in the file to show the layout of the desired source code file. It uses tubes to connect two matched nodes in two successive versions. To follow the entire evolution of a particular code fragment for example, a code swap can be easily detected by crossing tubes. The visualization techniques are implemented using the Tulip visualization framework.

## 2.3 Time-Series Visualization

Time-series data can be visualized in many ways. The most important visualization techniques for time-series data are sequence charts, point charts, bar charts, line graphs, and circle graphs.

### 2.3.1 Animation vs. Static

Dynamic hierarchies can be visualized in two ways, either by animation or by presenting the change using successive diagrams. The drawback using animation to visualize dynamic hierarchies is that the core diagram gets lost and we can't keep track of what exactly has changed, especially when we need to keep track of data in various timesteps. The mental map can therefore be lost which may lead to misinterpretations, as will be discussed thoroughly in the next section. Examples of such hierarchies using animation to represent dynamic hierarchies, are information slices and InterRing as were explained in details in Section 2.1.



Static hierarchies on the other hand can keep track of all changes during different time intervals, by displaying the different hierarchies successively against each other. Colors or lines can be added to visualize the changes. This remedies the misinterpretation and preserves the original hierarchy structures in all times.

In our project, static hierarchies are used to visualize the dynamic hierarchies over time and lines are used with different colour codes to visualize the changes.

### 2.3.2 Mental Map

Since humans easily memorize pictures, graphs, maps, etc. basically anything that is sketched or visualized, so information represented in any of these formats can be easily retrieved from the brain. People usually describe a location of a given place for example from a virtual graph they visualize in their heads<sup>3</sup>, the person's perception of that image is known as "mental map"<sup>4</sup>.

In dynamic hierarchies, rearrangements of some nodes can occur, as well as the removal and addition of new nodes [20]. For example an added node can overlap an existing node which can also effect the reposition of other nodes, this can effect the user's mental map on the original diagram structure. The mental map of the user should be preserved for ease of understanding on how the hierarchy structure has changed over time [11]. Otherwise, if the mental map was lost the viewer would take one object for the other over time and consequently makes misinterpretations.

In this project we visualize dynamic hierarchies while preserving the mental map [10], by implementing a static diagram and illustrating the changes applied to hierarchies like added, removed or moved nodes, using coloured lines, maintaining the basic hierarchy structure. Lines demonstrate the change by having its start point where the node was located in the previous hierarchy, and its end point where it is located in the successor hierarchy. Lines are coloured to express different meanings; "red" is to represent removed nodes, "green" to represent added nodes and "blue" to express moved nodes, see an illustration for clarification in Figure 2.16.

---

<sup>3</sup>[http://www.fedstats.gov/kids/mapstats/concepts\\_mentalmaps.html](http://www.fedstats.gov/kids/mapstats/concepts_mentalmaps.html)

<sup>4</sup>A mental map is an individual's own internal map (person's personal point-of-view perception) of their known world. <http://geography.about.com/cs/culturalgeography/a/mentalmaps.htm>



**Figure 2.16:** IPTPs with coloured lines between successive hierarchies representing the changes

# Chapter 3

## Case Studies

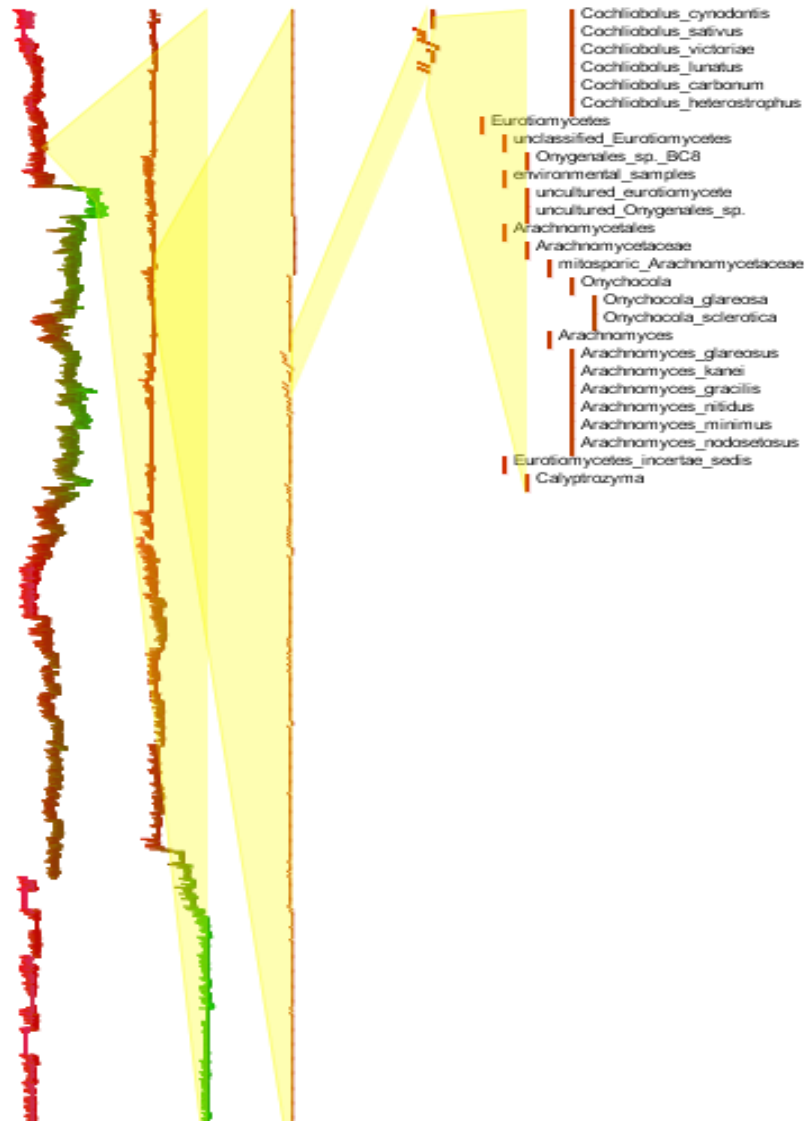
To illustrate and test our application and demonstrate the usefulness of IPTPs, we applied the IPTP technique to several datasets. We investigate very large datasets such as the National Center for Biotechnology Information (NCBI) taxonomy that contains several hundred thousand nodes (324,000 nodes) representing the names of all organisms that are represented in the NCBI genetic databases with at least one nucleotide or protein sequence [24]. This huge file was understandable from the IPTP representation, which may be difficult when the same dataset is visualized in a treemap, a layered icicle plot, or a node-link diagram due to visual scalability reasons. Furthermore, the plot can be scaled down immensely and the hierarchical structure still remains visible. Figure 3.1 illustrates the IPTP representing the NCBI taxonomy dataset where subhierarchy selection is applied - shown by the yellow triangles - until reaching a detailed view. From the overall view of the dataset represented in Figure 3.2, we can easily detect the deepest part of the IPTP as highlighted by the blue rectangle for example. This is easily grasped since we know that green nodes represent the leaf nodes and the higher the depth level it gets, the more it is indented to the right.

The only problem with the NCBI dataset is the long time it takes for the IPTPs to get plotted and to adopt to whatever action is taken on them.

Another kind of datasets were used for testing and demonstrating dynamic hierarchies. They represent the evolving prefix tree structure of words occurring in paper titles. All papers from the field of computer science are collected for each year. Then these are preprocessed to generate a prefix tree for each year. The data comes from the DataBase systems and Logic Programming (DBLP)<sup>1</sup> and is given in an XML file.

---

<sup>1</sup>DBLP server provides bibliographic information on major computer science journals and proceedings.



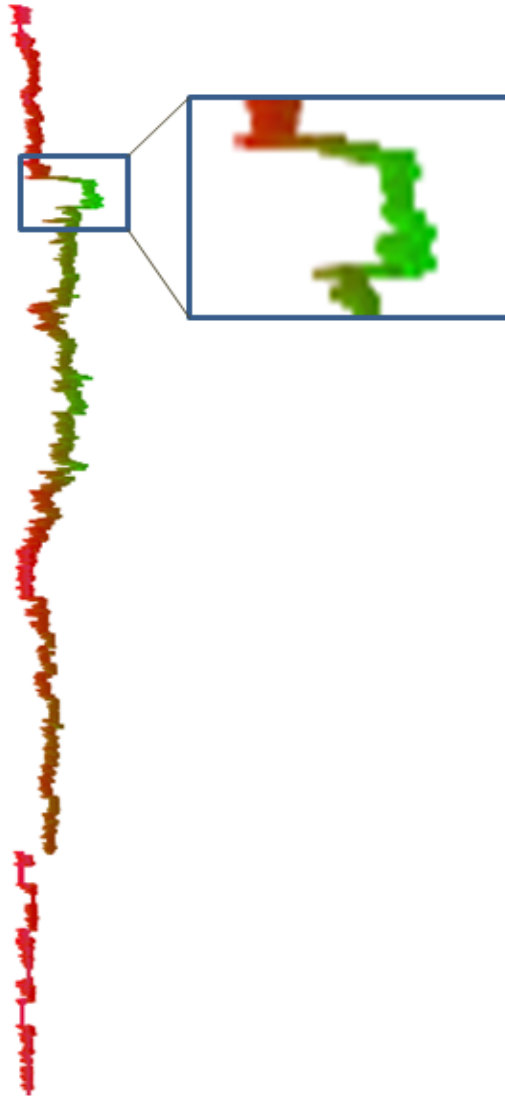
**Figure 3.1:** IPTP representing the 'NCBI taxonomy' highlighting a subregion

Two DBLP datasets were tested, a small DBLP consisting of 100 elements with a maximum depth of 4 'dblp.newick.100', meaning it is not very deep but it consists of many timesteps. The other dataset is bigger than the first one with up to 1,000 elements with a maximum depth of 4, 'dblp.newick.1000'. The hierarchies start with small numbers of elements and get bigger over time. For example, in the year 1949, there are only 145 elements and in 1974 the size of the file reached 1,000 elements.

All datasets are represented in newick file format<sup>2</sup>, which will be explained later in

---

<sup>2</sup>Newick tree format is a way of representing graph-theoretical trees with edge lengths using parentheses and commas.



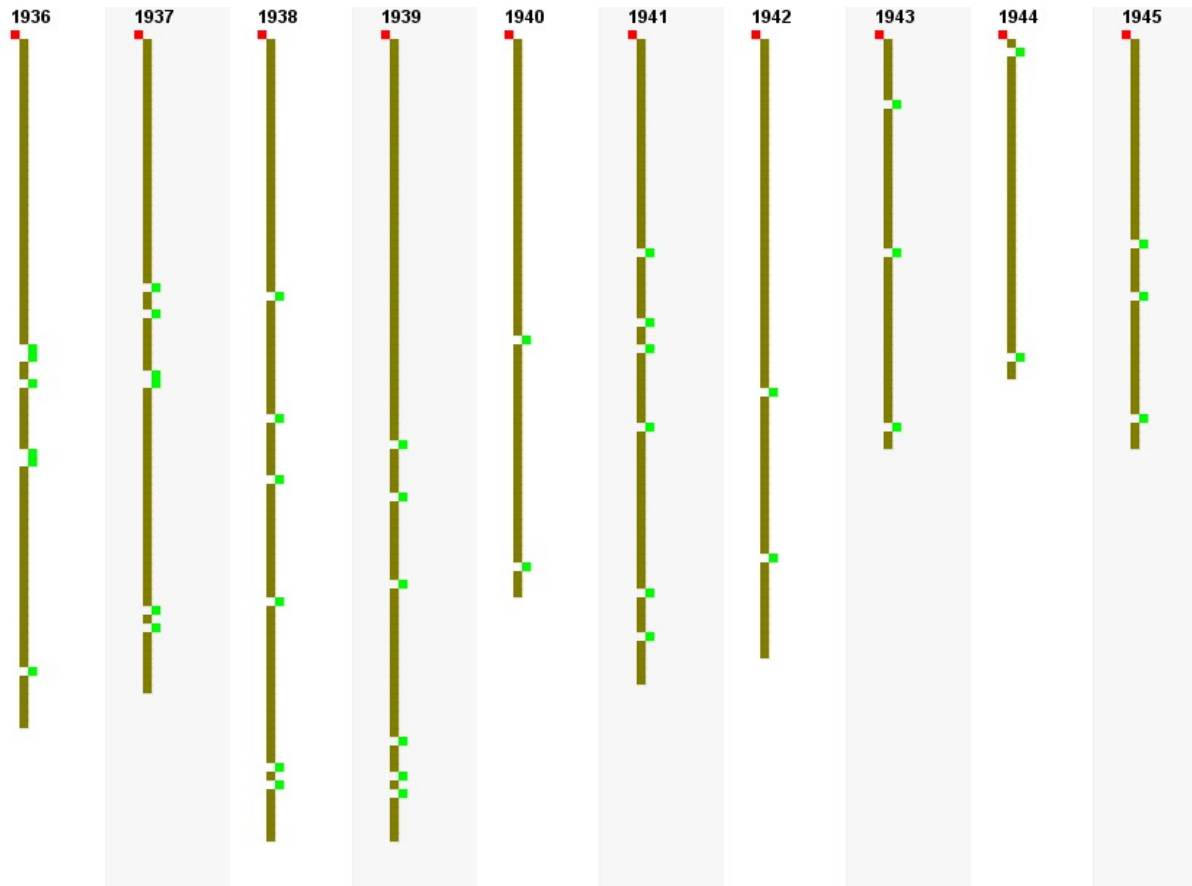
**Figure 3.2:** IPTP representing the 'NCBI taxonomy' illustrating the deepest leaf nodes (enclosed by the blue rectangle)

Chapter 6. The evaluation of the tool with all its features and functionalities were done on the 'dblp.newick.100' file as will be discussed.

The dataset in use represents data from 1936 until 2012. Since the display area can only take 10 IPTPs, analysis and evaluation will be discussed on a 10 IPTP view.

Figure 3.3 represents IPTPs from 1936 until 1945. Depth can easily be depicted by the different colors given to each depth level as can be observed by the Hierarchy Legend as shown in Figure 3.4

After comparing the IPTPs displayed as shown in Figure 3.5 we start analyzing how hierarchies evolved over time using the coloured lines knowing that red lines represent re-



**Figure 3.3:** IPTPs of the 'dblp.newick.100' file

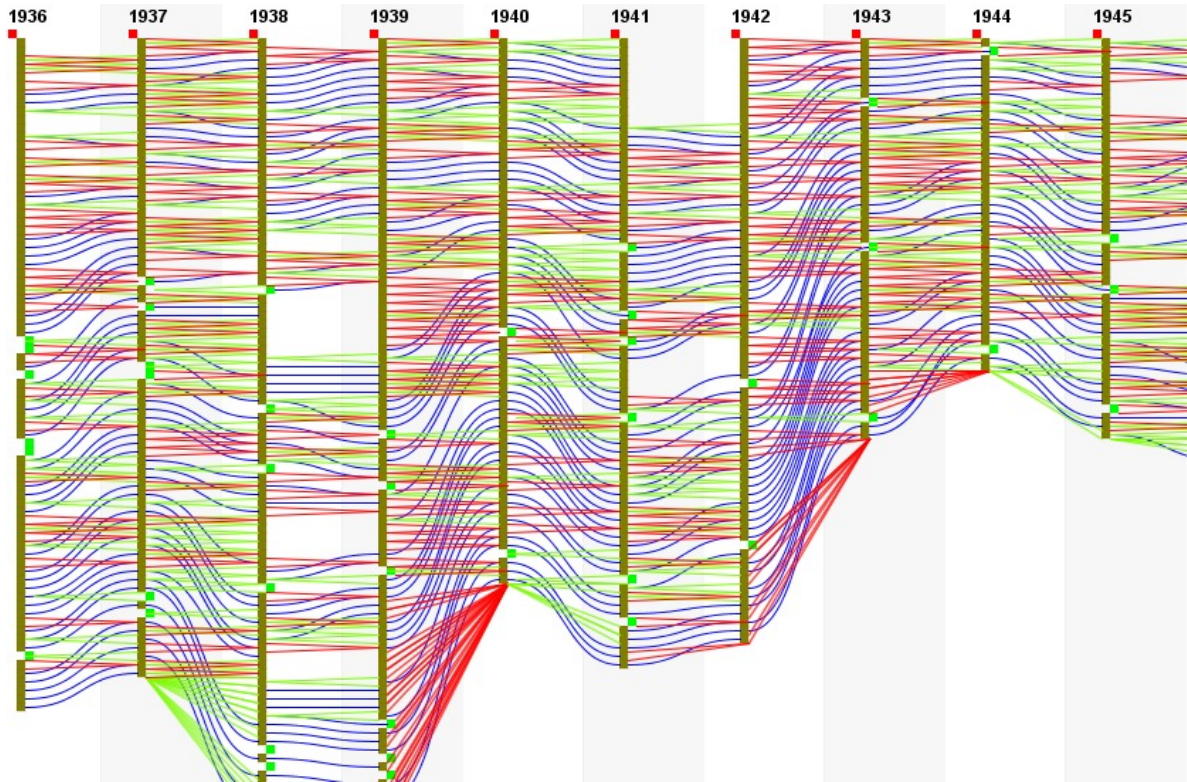


**Figure 3.4:** Hierarchy Legend

moved nodes, green lines represent added nodes and blue lines represent changed position nodes.

To better understand the changes, we illustrate some of the tool's features like filtering and searching and analyze our observation. Figure 3.6 represents the IPTPs after applying 2 filters, the first is to filter only nodes starting with alphabet 'a' and the second is to filter nodes starting with alphabet 'p'. From the observation, we can clearly see that from year 1941 to year 1942 there wasn't any removed, added or changing position nodes. As for the nodes starting with the alphabet 'p', we can see that from 1944 to 1945 the nodes only changed positions in the hierarchy.

To have a look at a specific node and observe the changes, 'Searching' is applied. For example in Figure 3.7 we can observe and keep track of changes on a node called



**Figure 3.5:** IPTPs of the 'dblp.newick.100' file after comparison

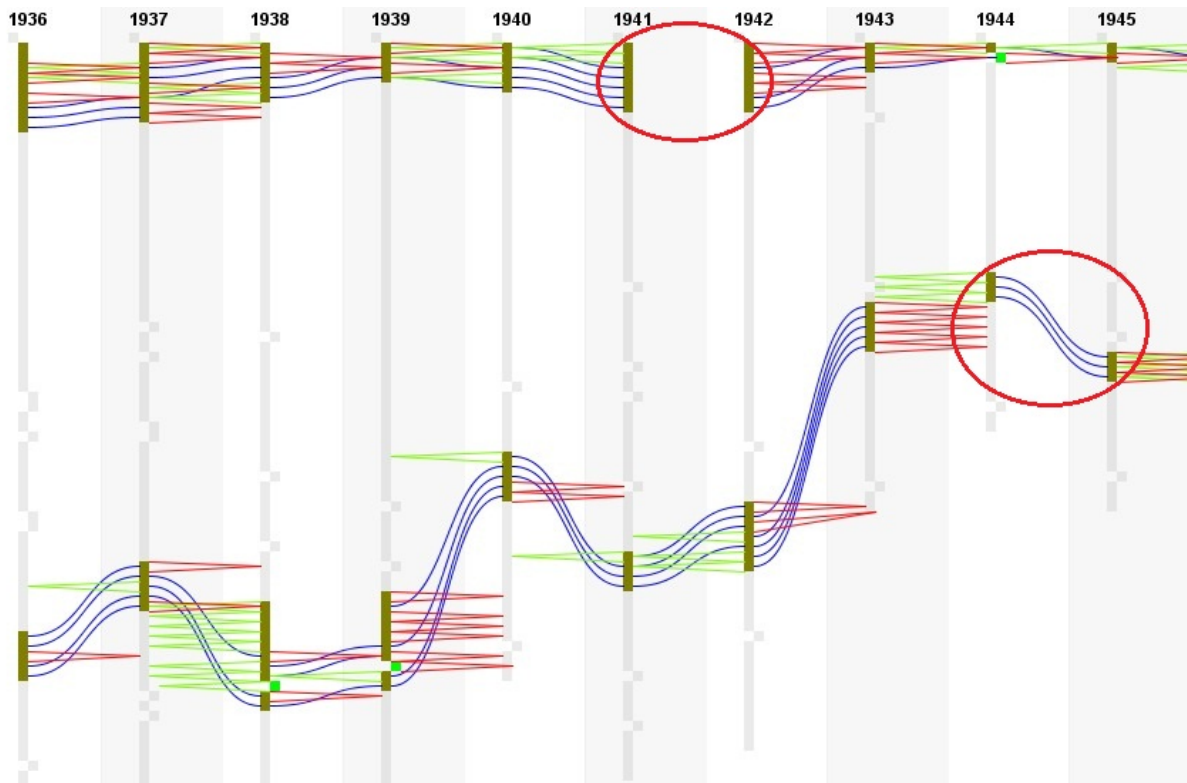
'boolean'. We can see that it was added in 1937 and removed in 1939, then reappeared in 1945.

Knowing that the system can display a large number of files consecutively, in order to gain insight on specific hierarchies at certain timestamps, we use the time filter to choose the hierarchies we are in need to analyze. We give a start date and an end date of hierarchies to display. Figure 3.8 shows IPTPs starting from 1940 until 1948 according to the user selection of dates. So instead of searching all along the huge number of files, time filter as shown makes analysis easier when in need of observing changes at specific timings.

Keeping track of only added nodes or removed nodes is also an easy task. Predefined filters are implemented and can be chosen from a drop down list. Figure 3.9 is an example of IPTPs after applying the time filter and choosing to only show added nodes (represented using green lines) on only nodes starting with the character 'a' as well as on the node named "arithmetic" (represented in yellow). As can be observed from Figure 3.9, in years 1942 and 1943 neither node starting with alphabet 'a' nor 'arithmetic' is newly added. Observing the IPTPs we can analyze that since 1940 'arithmetic' was added in 1945, changed position in 1946, removed in 1947, and reappeared in 1948.

Distinguishing nodes and displaying their names can be done by hovering over the





**Figure 3.6:** IPTPs of the 'dblp.newick.100' file after applying filters

nodes which then displays a tooltip with the node name. As seen in Figure 3.9, "axiom" is the node hovered.

The analyses above are all made by IPTPs observation and by applying different types of filters for better investigation. Other analyses can be made by observing the statistics displayed - as seen in Figure 3.10 - after IPTPs selection, which demonstrates the changes in percentage between IPTPs, like how many nodes were added, removed or changed positions. This is shown by the red rectangle in Figure 3.10, which illustrates the changes between IPTPs of years 1936 and 1937: 26% of nodes were added, 31% were removed, and 52% changed positions in 1937. The Figure also highlights another blue rectangle that demonstrates file details of the year 1936, we can see that it contains 80 nodes and a maximum depth level of 4. This information changes accordingly whenever an IPTP is selected.



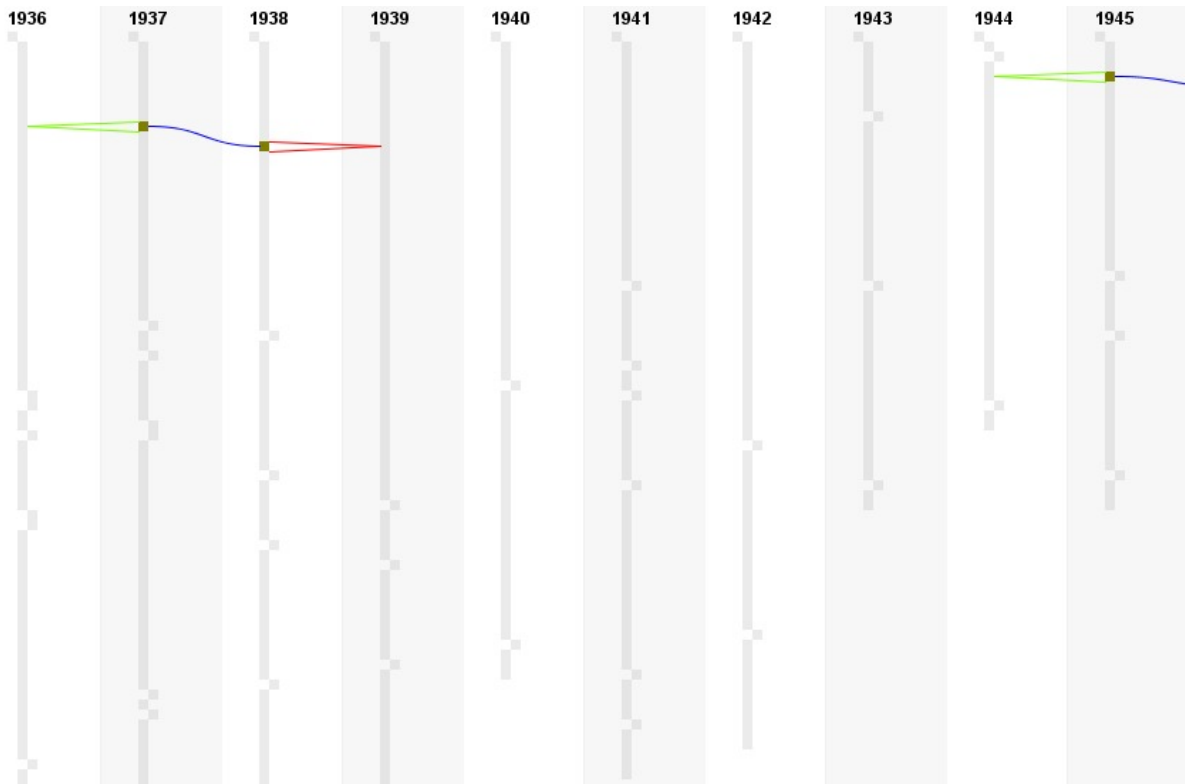


Figure 3.7: IPTPs of the 'dblp.newick.100' file after searching the element 'boolean'

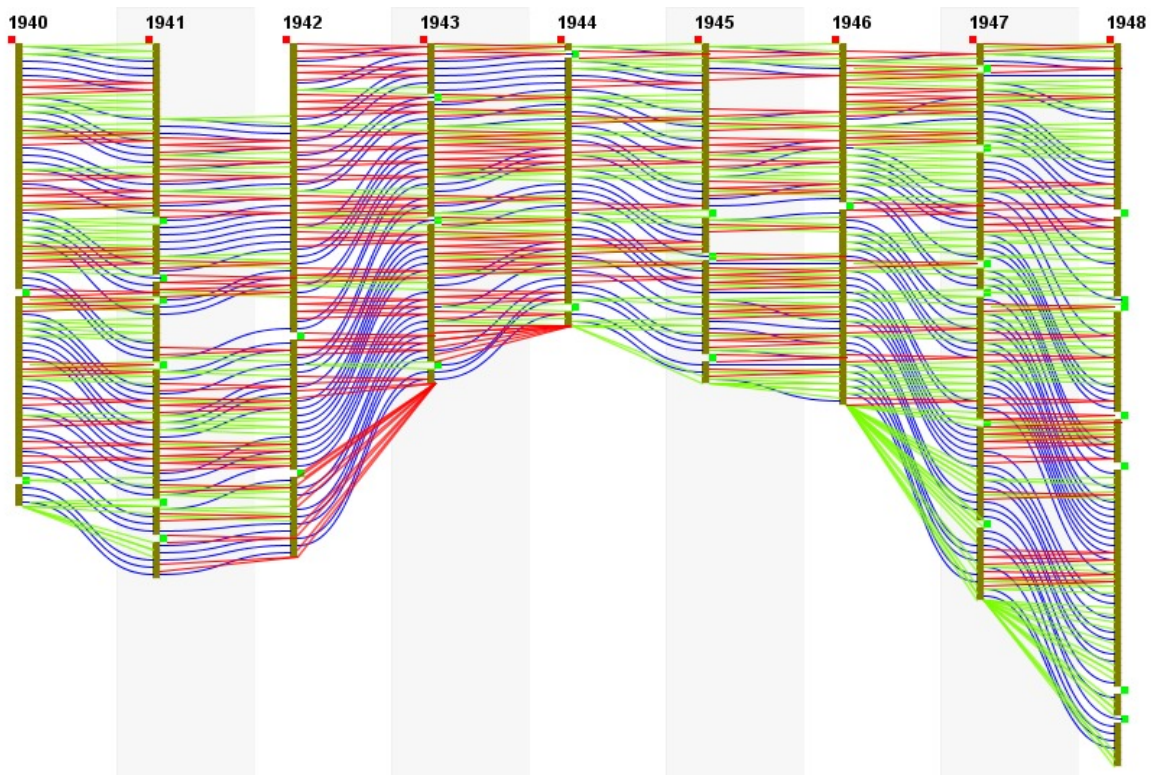
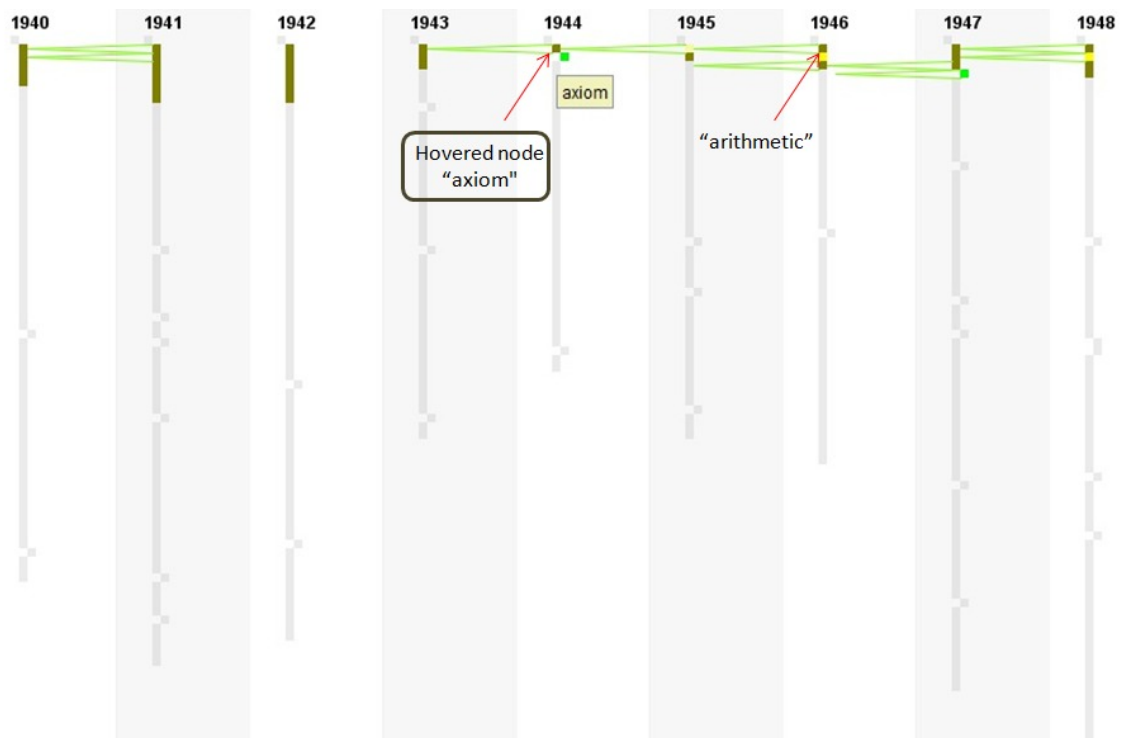
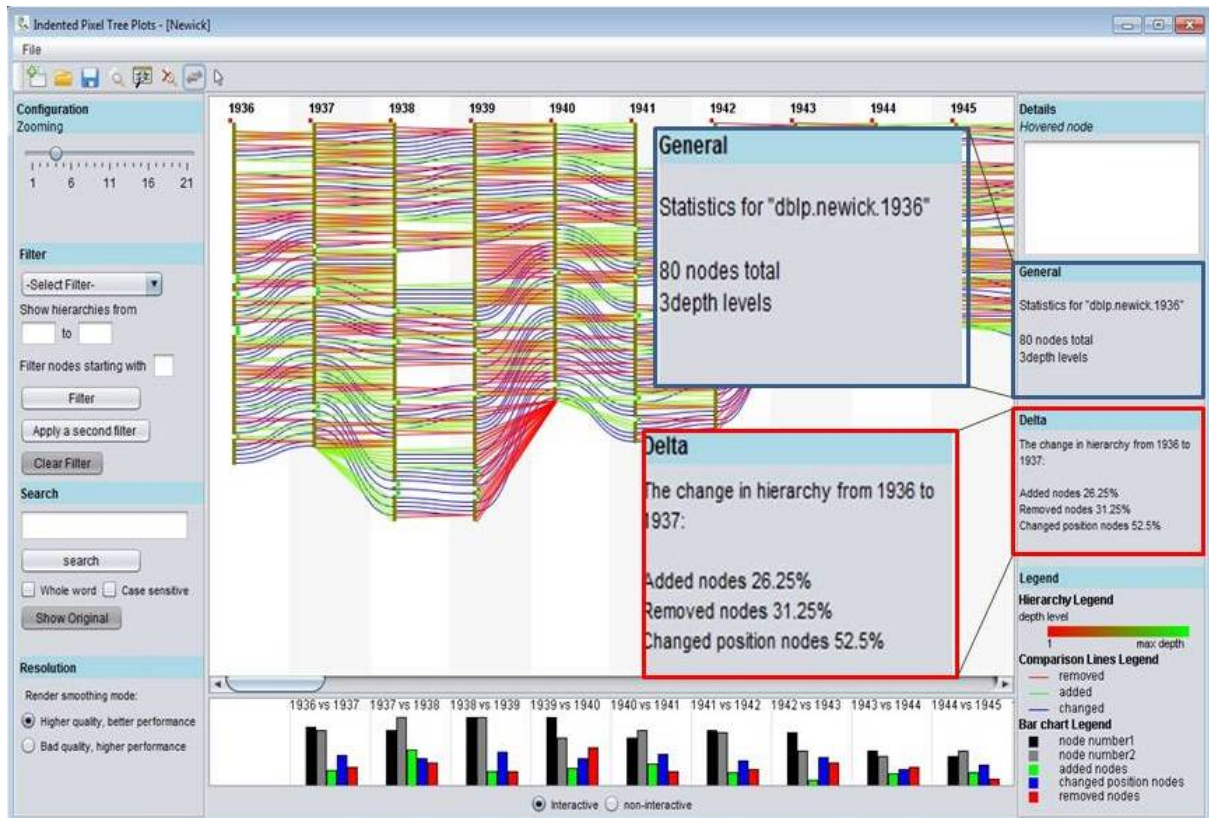


Figure 3.8: IPTPs of the 'dblp.newick.100' file after applying the time filter



**Figure 3.9:** IPTPs of the 'dblp.newick.100' file after applying the time filter, and show only alphabet starting with 'a' as well as the result of a search on "arithmetic" (shown in yellow)



**Figure 3.10:** General information about IPTP of 1936 (blue rectangle) and changes between IPTPs of years 1936 and 1937 (red rectangle)

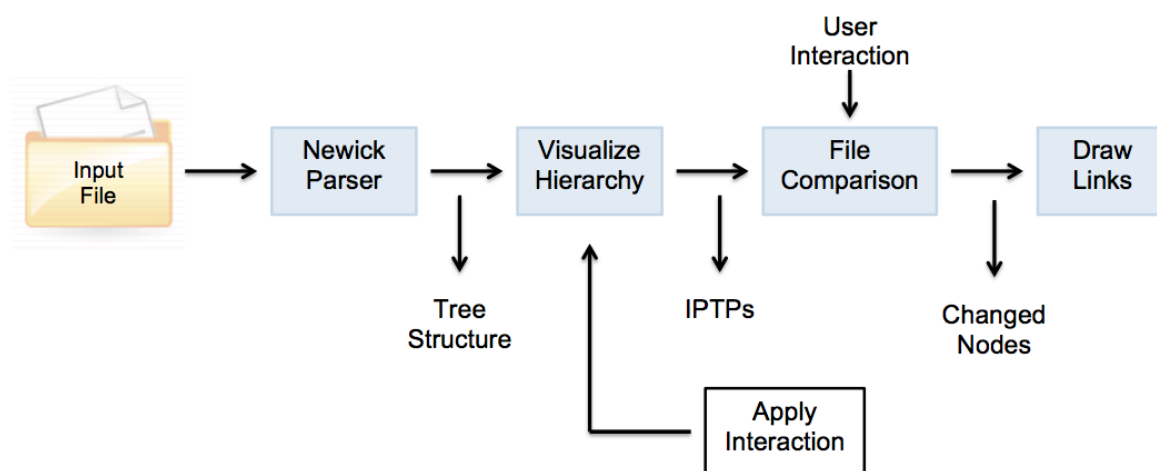


# Chapter 4

## Project Architecture

### 4.1 Process Overview

This section gives an insight about the process of visualizing IPTPs and how they evolve over time. This process can be sub-divided to multiple steps as can be viewed in Figure 4.1.



**Figure 4.1:** Process overview diagram

First, we use a newick file parser to extract a hierarchical tree structure  $T_i$  consisting of nodes from every newick file version  $f_i$  of the files contained in the directory of interest. Second, we display the hierarchies (IPTPs) corresponding to the parsed files. Third, a comparison takes place and we use the "Comparison Algorithm" described thoroughly in Section 6.4 to detect the changes between consecutive hierarchical structures  $T_i, T_i + 1$ . Fourth, we draw links between consecutive hierarchy nodes to represent and illustrate the

changes calculated in the previous step. Many interactive features can then be applied on the hierarchies visualized for a better understanding, discussed in Section 5.3 where we will describe each step in detail and how they were implemented in Chapter 6.

## 4.2 Class Diagram

In order to best illustrate the interdependencies and relationships among the different system modules, we thought it was best practice to create a class diagram. The here-under depicted Figures 4.2 and 4.3 define the overall static view of the system, where a description of how each class (together with it's associated attributes) relate to every other class present in the system.



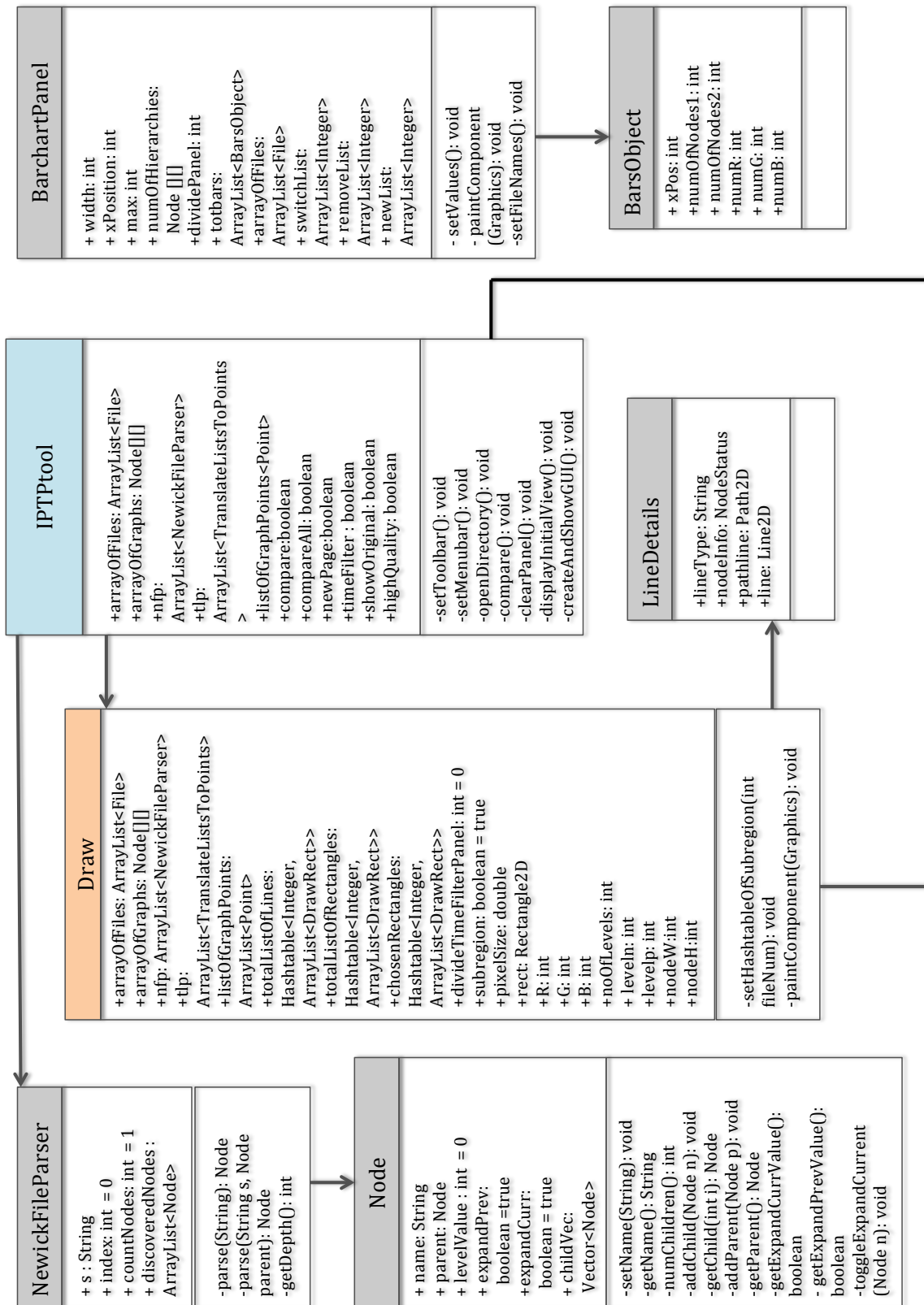


Figure 4.2: Class diagram (1)



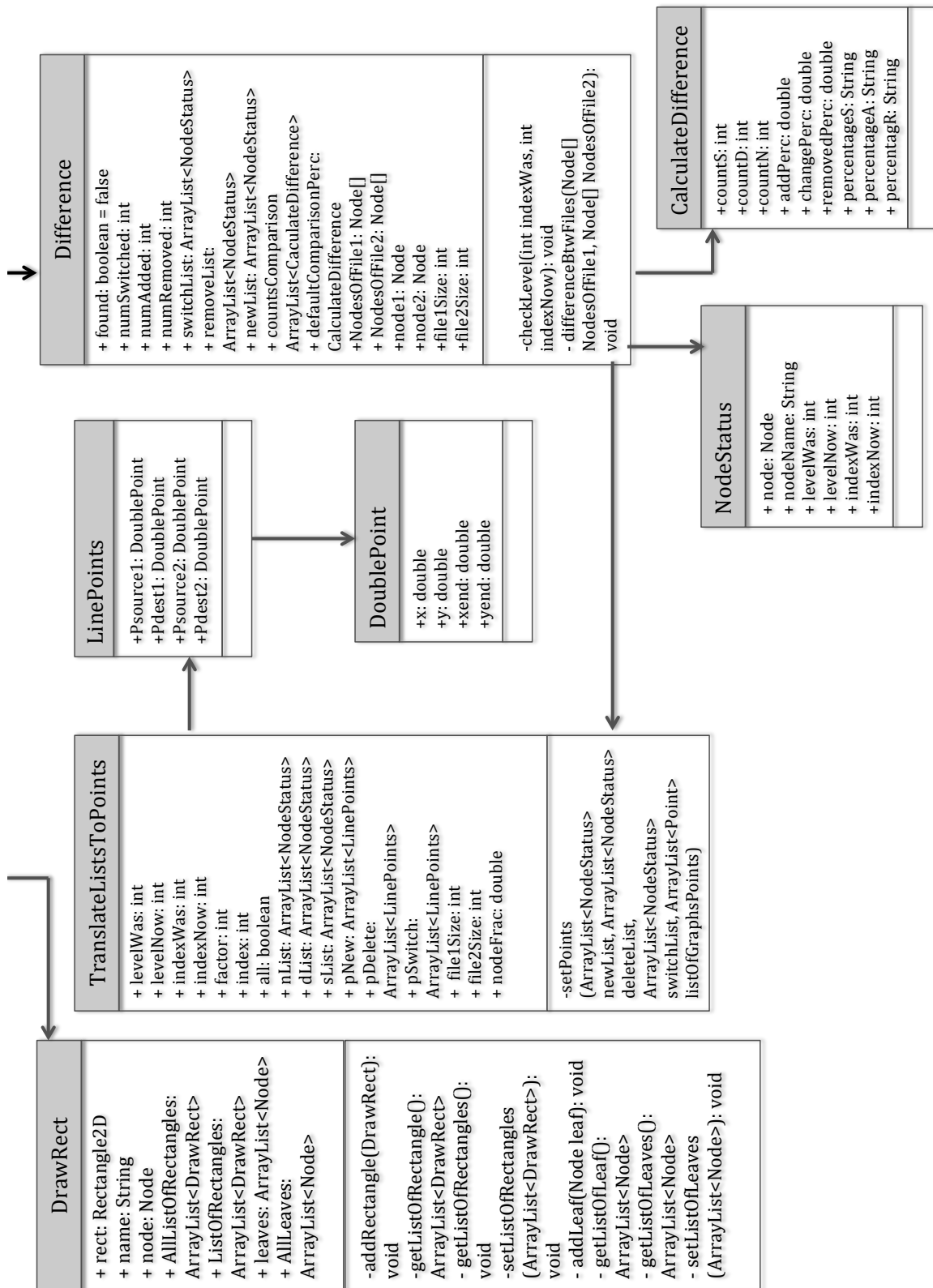


Figure 4.3: Class diagram (2)

## 4.3 Features and Functionalities

Navigation and interaction facilities are essential in Information Visualization. Our tool is based on the visual information-seeking mantra: overview first, zoom and filter, then details-on-demand [26].

The tool supports a variety of interactive features to explore the hierarchical data such as collapsing and expanding specific nodes, getting information on a specific element and selecting a subregion for a hierarchy on a larger scale for better investigation. The tool's features and functionalities are as follows:

- **Region selection:** Part of the indented plot can be selected by mouse pressed and mouse released functionality, highlighting a desired region. The selected region is then displayed at the right of the original hierarchy within a larger scale for a better observation.
- **Hierarchy expanding/collapsing:** Using the mouse click functionality, clicking on a node representing an attribute can collapse or expand its corresponding children alternatively. So when a node is collapsed all its sub-nodes (children) are hidden, and whenever the node is expanded again the sub-nodes reappear.
- **Text pattern search:** Typing in certain text in a searching browser, searches for the specified text in the indented plots displayed and highlights the corresponding nodes found.
- **Zooming:** A zoom in/out horizontal slider bar can be used for a larger/smaller scale on the indented plots displayed on the display screen for a more detailed representation or an overall view alternatively.
- **Details-on-demand:** Using a mouse over functionality, by moving the cursor on any element on the display screen, information is displayed as a tooltip at the current mouse cursor position. A tooltip could be placed either on a node element or a comparison line. A more detailed information for a specific element (node) like the node name, number of children of the node, the parent name and the depth is displayed as well on a text area shown on the tool's right panel.
- **High quality/Low quality (Resolution):** For a faster rendering specially when representing huge datasets, the user can choose whether to display the indented plots with high quality (better resolution) with no aliasing or to have a bad quality plots rendered with aliasing.

- **Interactive/Non-interactive (Display mode):** The display mode can change whenever needed to either interactive or non-interactive mode. Meaning, if only a static view of the indented plots is needed with no interaction then the user can choose to switch to a non-interactive display mode which makes rendering faster. The user can change the display mode to interactive whenever needed. A static view is of importance when displaying huge hierarchies when there is no need of the interactive features, like only having the ability to scroll horizontally and vertically.
- **Filter:** A list with some pre-defined filters is available on the tool's left panel where the user can choose certain filter function like displaying only nodes that got added, on further hierarchies, or displaying only nodes starting with certain alphabet like 'a' for example.
- **Color coding:** For a better visualisation of the different hierarchy levels, each level of nodes is represented in a different colour to distinguish it from other elements having other level values. So by user observation of the different colour gradients, user can easily identify the deeper levels from other shallow levels.
- **Comparison:** A comparison button is provided on the tool bar to provide and illustrate changes between consecutive hierarchies displayed. Colored lines are used to illustrate changes from one hierarchy to another.
- **Delta (Percentage of change):** Percentage of change on added, removed and changed position nodes between two consecutive hierarchies selected can be displayed.



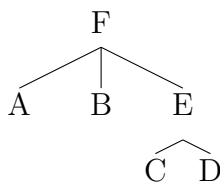
# Chapter 5

## Implementation

The following chapter gives a full explanation for the main building blocks of the tool. In order to help illustrate the ideas, additional descriptive elements including equations, figures and pseudocodes were also used when needed. The chapter starts by introducing the *Newick File Parser*, which is responsible for the transformation and building of the tree hierarchy from an input given as string. The *Collapse/Expand Algorithm* section describes how the node expansion and collapse is performed. Comparison algorithms are discussed in details next in the following chapter *File Hierarchies Comparison*. The visualization module is outlined in the subsequent section. The chapter encloses with the *IPTPtool* section where the main method and GUI modules are implemented.

### 5.1 Newick File Parser

The main goal of this class is to parse the input file, that is given in Newick format. The "Newick" format uses parentheses and commas to show the parent-child relationship to represent a hierarchy of data, e.g.  $(A,B,(C,D)E)F;$ . The tree representation of the newick file is shown in Figure 5.1.



**Figure 5.1:** Tree representation of the parsed newick file  $(A,B,(C,D)E)F;$

As the edges don't play a role in our file representation no processing on the edges encountered during the traversal of the newick file is performed. The `NewickFileParser` is implemented to traverse the parents and their children and record them in an `ArrayList`

'discoveredNodes' used for further processing throughout the application.

The file is read and stored in a `String newickString` that is then passed to the parser. The parser uses a recursive method to explore all the nodes. First, the parser extracts the root node knowing that it is placed at the end of the `newickString` before the semicolon ';', and stores the rest of the `newickString` representing the root's children. So in the example stated above, the root node will be F and its children will be A,B,(C,D)E. The parser then calls the recursive method taking as parameters the 'Parent' node (the root in this situation) and the 'newickString' representing its children.

Before describing how the input string is processed and passed recursively to the method, the notion of bracket balancing and its benefits needs to be explained first. Bracket balancing simply aims at detecting the direct child nodes of the current parent nodes, and extracting the substring representing their sub-child nodes. In other words, the first direct child node to the current parent node is the first string after a perfectly balanced bracketing. The second direct child node to the current parent node is the first string after the second perfectly balanced bracketing, etc. Every direct child node detected is then recursively passed as parameter together with the preceding perfectly balanced substring to the same node to discover the nested child nodes. In order to achieve this, the input string is traversed from left to right character by character inspecting each one to determine its state. The perfect bracket balancing is being tracked by a variable called `bracketcount`. `bracketcount` is always incremented by 1 if an open bracket is encountered '('). In contrast it is decremented by 1 if a closed bracket is encountered ')'. Any substring that is traversed while the `bracketcount` has a greater value than zero, is simply concatenated to form the string to be passed for the next recursive call. The first substring encountered while the `bracketcount` equals zero is treated as a direct child node for the current parent node, and passed recursively as a parent for the preceding concatenated substring (i.e. the substring between the preceding balanced brackets).

Before creating a new node the depth value of the node needs to be set. The root node is first initialised with a node depth value of 1 accordingly, all other nodes' levels are set. If the new node being created is a child to the previous node created, then the depth level value associated with the previous node (the parent node) is incremented by one and assigned to the depth level value of the new node.

The algorithm repeats until the whole `newickString` is read. The depth of the whole

tree can then be calculated using the `getDepth()` method that gets the maximum node level stored from the `discoveredNode` list containing all created nodes resulting from the parsed `newickString`.

After parsing the newick file, the tree structure is then perceived. Since we are in need of an indented outline for hierarchy visualization, our implementation is a kind of a depth first search traversal.

```

parse(String s, Node root)
  //s consists of only one child(one Node)
  if is-simple-struct(s) then
    | return new Node(s);
  end
  //complex structure
  if lis-simple-struct(s) then
    String buffer=" "; //to store the String representing a Node's children
    for int i=0;i<s.length;i++ do
      //when bracket count reaches zero
      if is-balanced(charAt(i)) then
        | Node p=new Node(charAt(i));
        | root.addchild(p); //add node 'p' as child to the parent 'root'
        | set-node-level(p); //depth level of node 'p' is set to be equal to the
        | parent node level+1
        | if buffer is not empty then
        | | remove-outer-brackets(buffer);
        | | p.addchild(parse(buffer,p));
        | | buffer=" ";
        | end
      end
    end
    else
    | buffer=buffer+charAt(i); //if unbalanced buffer it up
    end
  end
end

```

**Algorithm 1:** Newick file parser

### 5.1.1 Node

A tree structure is the ideal representation for any hierarchical data structure. It is represented as a collection of nodes to represent the attributes. Each node is a data structure consisting of a value. In our application 'Node' is the data structure used, consisting of a **String** value which is the attribute name. With each node a record of its parent, list of children and a **boolean** representing its collapse/expand state are saved, along with the attribute depth level. Nodes are distinguished and known using a name of type **String** as their unique key.

The parameters of this data structure are as follows:

- \* **String name**: the name of the node  
**Node parent**: the parent node of the current node
- \* **levelvalue**: the depth value of the node
- \* **Vector<Node> childVec**: a vector to store the node children
- \* **Boolean expandcurr**: is set to **true** to initialize all nodes to be expanded. If this is set to **true** this node is collapsed, meaning all its children and all its descendants are hidden.
- \* **Boolean expandprev**: Initialised with a **true** value. This **Boolean** is to indicate that the parent of that child is collapsed. It is set to **false** if **expandcurr** equals **false**. It is set for all the descendants of any collapsed node (having its **expandcurr** equals **false**).

## 5.2 Search Engine

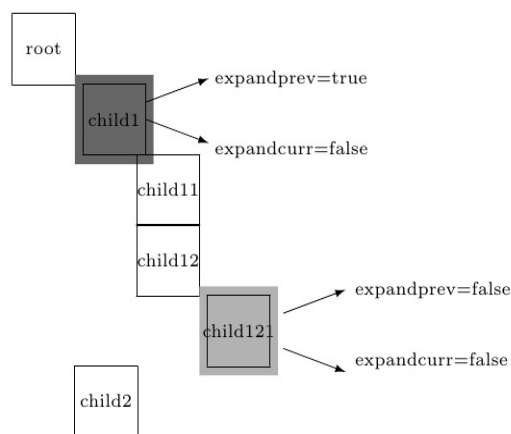
When the search button is clicked, **SearchEngine** class is called, taking as parameters the **String** written in the text field. Searching is done on a 2D array containing all nodes consisting the IPTPs on the display panel. Depending on the preferences of the user, whether search on the text is done on the whole word and/or is case sensitive, the searching checks differ. Two booleans are used to check whether the word options are selected or not, and this is how we differentiate between four checks. The four checks are a combination of both booleans. If "case sensitive" is unchecked, then the search is done using an Ignore case comparison. If "whole word" is checked, then the **String** entered is only compared to a substring of the node names in the array we are searching in. A data



structure, `SearchFound`, is used to store nodes in case the search was successful for further visualization purposes. `SearchFound` takes the node name, the file index containing it and the node index in the hierarchy as parameters.

To visualize the effect of the searching process in case of a successful search, a Boolean `SearchFilter` is set to `true`. This is done so that at the painting process, the list of `SearchFound` data type is checked before drawing each node, if node is found in the list then it is drawn with its colour code (depending on the node's depth level) otherwise, it is drawn with a grey colour. In case the node searched for is already highlighted by another filter, the node is then highlighted in yellow to distinguish it from other visualized nodes.

### 5.3 Collapse/Expand Algorithm



**Figure 5.2:** Collapse/Expand illustration

Collapsing and expanding nodes is achieved using two `boolean` variables, one to indicate that this node is collapsed and another to state that the parent of this node is collapsed:

- `expandcurr` this variable if `false` states that this node is collapsed, meaning all its children and descendants are hidden.
- `expandprev` when `false`, this is to state that the parent of the current node is collapsed. We need this variable in our implementation so that when any descendant of a collapsed node is getting drawn it checks first if its parent is collapsed (`expandprev` is set to `false`) if it is, then this node is also hidden.

A node is collapsed/expanded when a rectangle representing a node is clicked, we check its position and compare it with the positions of all nodes representing the hierarchies displayed. If node is detected it is then stored for further processing. This node is then toggled changing the `expandcurr` value from `true` to `false` or vice versa. This is done by calling the `toggleExpandcurrent` method in the `Node` data structure.

When drawing the nodes in the hierarchy, if the parent node of the node being drawn has `expandcurr` equal to `false` or the node being drawn has `expandprev` equal to `false`, then this node is skipped from drawing.

To visualize the collapsed node, a `boolean collapsed` is set to `true` when the node's `expandcurr` value is set to `false`. When this `boolean` is checked to have a `true` value, this node is drawn with a shaded colour.

```

initialization;
if node is pressed then
    toggle the node's expandcurr value;
    if expandcurr of parent node=false or expandprev of parent node=false
    then expandprev of node=false;
    continue;
    if expandcurr of node=false and node has children then set boolean
    collapsed to true;
end

```

**Algorithm 2:** Collapse/Expand

## 5.4 File Hierarchies Comparison

Comparing file hierarchies, displays information on what nodes have been changed, removed or added. In our tool the comparison of hierarchies' implementation, has two applications; either we apply the comparison on all displayed hierarchies, or compare hierarchies depending on the user's selection.

Different coloured lines are used to visualize the changes between hierarchies. Stated below is a brief explanation on the comparison algorithm and how each of the different application stated above is implemented.

### 5.4.1 Comparison Algorithm

Displayed hierarchies on the tool's screen, are all stored in a list. A "compare all" or a "compare specific hierarchies" button shown on the tool bar should be pressed for the comparison to take place. Pressing a button calls a compare method, which by its turn

makes calls to two main classes, *Difference* and *TranslateListsToPoints*. Class *Difference*, takes two successive hierarchies of nodes; *h1* and *h2*, and stores the nodes that have been removed, added and changed in three different lists. To define what nodes were deleted, changed or added, nodes are first compared one by one by their names, comparing *h1* with *h2*. If a node in *h1* is found to be also in *h2*, then we know it has not been removed or added. Next step is defining whether it has changed its place or not. This is checked by comparing the nodes' levels. If levels are not equal, we know it has changed places, if not, then we need to do further checks to see if the node changed place between siblings.

If the names to be compared were never matched, then we know this node has been removed. To define the added nodes, we change the order of hierarchies to be compared; which means we take *h2*, compare it with *h1*. If we find that this node is not matched with any node in *h1*, then this is a newly added node.

The second class takes the three lists storing the added, removed and changed position nodes. These lists not only store the node name, but the level and position of the node as well. These details are needed in this class where we need to define the start and end point of each line.

The last class is the one responsible for drawing the comparison lines accordingly. Red lines represent deleted nodes, while blue lines are for nodes with changed positions, and finally, green lines which represent the added nodes.

## **Comparing Specific File Hierarchies**

The only difference between the two possibilities: comparing all or specific hierarchies, is the list containing the hierarchies to be compared. To be able to select specific hierarchies, a green transparent rectangle is drawn on top of the chosen hierarchy. The rectangle's bounds are being specified by the visualized hierarchy, depending on its level of depth. The chosen hierarchies are then stored in a different list, which is then passed to the first class instead of the list containing all displayed hierarchies, in order to perform the comparison.

## **Delta**

We not only represent the changes by drawing lines with different colour codes, but also represent them using numbers and charts.

Delta, is the name used to describe this paragraph as we will be talking about the changes that took place between different hierarchies, in different time intervals. A percentage of difference is calculated each time a comparison of two hierarchies is being executed

(as was mentioned in the above section, we compare hierarchies two by two). A list is used to carry all the percentage values for each comparison. The percentage of difference is calculated as follows: A class taking hierarchy 1 ( $h1$ ), hierarchy 2 ( $h2$ ), the removed nodes list, added nodes list and changed position nodes list as parameters is responsible for the calculation. To calculate for example the percentage of the added nodes in  $h2$  in comparison with  $h1$ , we divide the number of added nodes in  $h2$  by the size of  $h1$ . The Result of the calculation is then multiplied by 100 to get the value in percentage.

To have a better understanding on how hierarchy changes in time, it is always easier to have numbers visualized. Bar charts are used and drawn under each hierarchy displayed, representing the size of  $h1$  in black, the size of  $h2$  in grey, the removed nodes in red, the added nodes in green and the changed position nodes in blue. They represent statistics for two consecutive hierarchies. Figure 5.3 illustrates the bar charts in red rectangle and the percentages of change enclosed in the blue rectangle between IPTPs of years 1936 and 1937.

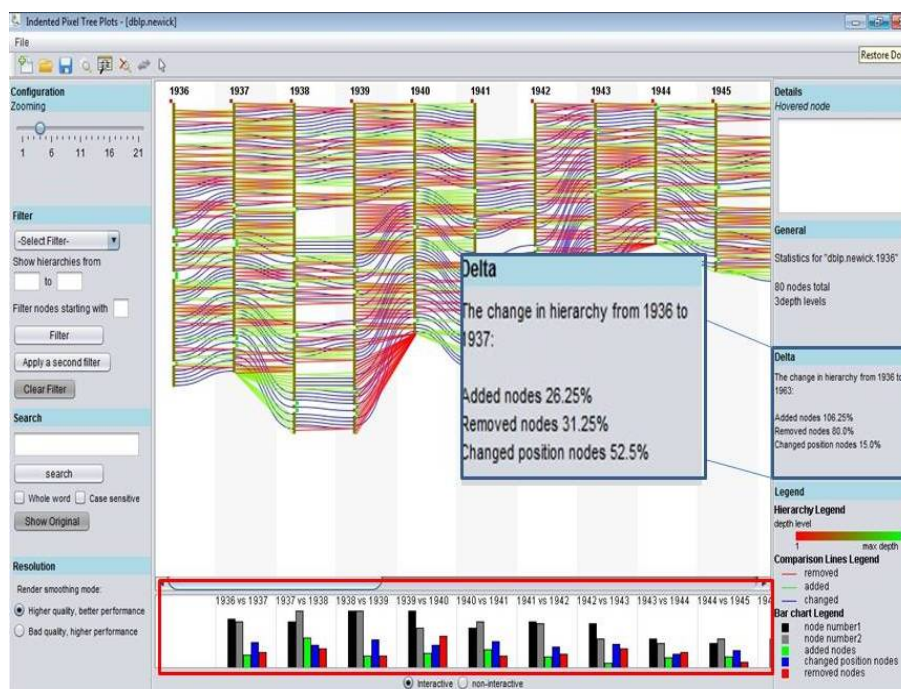


Figure 5.3: Delta

## NodeStatus

A data structure is created to store information about removed, added and changed nodes. To be able to track a node's status, whether it changed places, was added or got removed. This data structure stores for each changing node:

- **name** the name of the node
- **depth** the node depth in first hierarchy and the node depth in second hierarchy
- **index** the node index in first hierarchy and the node index in second hierarchy

The parameters are as follows: **Node**, **depth1**, **depth2**, **index1**, **index2**. According to the node status, - deleted, changed or added - some parameter's value change. If a node is added then **depth1** and **index1** get a value of -1. Same goes when a node is deleted, **depth2** and **index2** get a value of -1.

### **GraphBound**

In order to be able to record and observe the selected hierarchies when only specific hierarchies are needed for comparison, some parameters such as the starting point, **x**, **y** and the **width** and **height** of each selectable hierarchy should be stored. This is done by creating a new Object '**GraphBound**'. A **Rectangle2D** takes these parameters stated previously and creates the rectangle used to draw the bounds around the hierarchy. This is to be able to bound the hierarchy with a coloured rectangle to distinguish it from other unselected hierarchies.

### **DoublePoint**

**DoublePoint** Object is created to be used instead of the predefined Class **Point**, since a double precision is needed in visualizing the hierarchy. The need of presenting large datasets, urges us to draw the nodes in a way that makes a hierarchy composed of thousands of nodes to fit on the display screen. In order for that to work, we need to compress and overlap the nodes. A normal node will take up a size of a pixel; a rectangle having a width and height of size 1. So nodes can be drawn with less than a pixel size - a value between 0 and 1 - depending on the hierarchy size. Although, we are not in need of the double precision in all cases but it is used with all **Points** for ease of manipulation.

### **DrawRect**

The hierarchy is represented using a list of rectangles having a default width and height of size 1 (a pixel size). In order to store all nodes' positions for further processing, such as node selection, collapsing and expanding, a new data structure "DrawRect" is created taking as parameters:

- **Rectangle2D** to represent the node position
- **Node**

Nodes are represented and drawn using rectangles, for each node consisting the hierarchy.

## LinePoints

Each line displayed during the comparison process has specific points, a starting point and ending point to describe each line. To store and keep track of all lines that need to be displayed **LinePoints** is a data structure that is used to store the source and destination points of each line. This data structure is composed of **DoublePoint Psource** the starting point where the line begins and **DoublePoint Pdest** the end point of the line.

## LineDetails

A record for each comparison line representing removed, added and changed position nodes are stored using a **LineDetails** data structure. To be able to represent information on any line comparison between two hierarchies the **Line** or **Path** drawn should be stored along with the **NodeStatus** and the line type. For that to happen, this data structure consists of:

- **String** to represent the line type, which is either **delete**, **new** or **switch**
- **NodeStatus** to store all information associated with a node that changed status, explained in more details in the previous paragraph
- **Line** or **Path** depending on the line drawn since a line representing a changed position node is drawn using a **Path2D** data type and those representing an added or removed node is drawn using a **Line2D** data type

## BarchartPanel

For a better representation and understanding on how data is changing, a bar chart is displayed whenever a comparison of hierarchies is processed. The bar chart illustrates the number of removed, added and changed position nodes between each hierarchy with its previous hierarchy. The total number of nodes composing the hierarchies being compared (number of nodes in hierarchy 1 and number of nodes in hierarchy 2) are also being displayed. Bar charts are placed each under the hierarchy being compared to; e.g. hierarchy 2. Colours are used to distinguish between bars representing each attribute.

- **Black** represent the number of nodes of first hierarchy
- **Gray** represent the number of nodes of second hierarchy
- **Red** represent the number of nodes that got removed
- **Green** represent the number of nodes that were added
- **Blue** represent the number of nodes that changed places

## BarsObject

A data structure is used to store all attributes used to represent the bar chart representing the differences between successive hierarchies. For a bar chart to get displayed we need to keep track of several parameters such as the position of the bar chart (where on the Panel will it be displayed), the number of nodes in first hierarchy, the number of nodes in second hierarchy and the number of nodes in each of the following: removed, added and changed places lists. The following are the parameters of `BarsObject`:

- `x` the x position of the bar chart
- `numOfNodes1` the number of nodes representing the first hierarchy
- `numOfNodes2` the number of nodes representing the second hierarchy
- `numR` the number of nodes that got removed
- `numG` the number of nodes that were added
- `numB` the number of nodes that changed places

## 5.5 Visualization

Hierarchies describing the dataset of a given file in different timestamps, are represented and drawn using the *Graphics* library. For the hierarchies visualization, "Draw" class is implemented to handle any drawing processing needed in the whole application. As described in Chapter 1, hierarchies are visualized in an indented pixel tree plot style. We use a set of rectangular shapes to represent nodes contributing in the hierarchy construction. We plot the rectangles successively in an indented outline structure to form the desired hierarchy form.

Any changes in the hierarchies observed due to actions performed on the hierarchies displayed is also implemented in the "Draw" class. For example executing any filtering function on the hierarchies displayed, calls the paint method responsible for any drawing observed, which is found in "Draw". In order to distinguish each `repaint()` call when called from several parts of the code depending on the user requirement, control variables are used to drive the flow of execution of the different part of the `paintComponent` method blocks. For example when displaying new hierarchies to the display panel, a `boolean` variable `showOriginal` is set to `true`. Also when a filter function is applied, like a filter to highlight all attributes that got removed from one hierarchy to another, the `boolean` variable, `showRemoved` is set to `true` and all other control variables are set

to `false`.

What differs in each drawing is mainly the colors of each node (rectangle) and this depends on the control variables. Filtering is usually observed by highlighting specific nodes depending on the filter function, letting all desired nodes highlighted by their colour code, depending on their depth level and the rest of the nodes greyed out. So the main difference found between each drawing implementation in every condition is mainly the colouring of nodes. Nodes are coloured when needed, according to the filter function and greyed out otherwise.

### Hierarchy Visualization

The position of each hierarchy is decided depending on the number of hierarchies to be displayed on the display panel. These hierarchy positions are stored in a list `listOfGraphPoints`. This list is then used to draw each hierarchy in its specified position. This is how hierarchy positions are calculated:

$$\text{space taken by each hierarchy} = \frac{\text{display panel width}}{\text{number of files to be displayed}} \quad (5.1)$$

Then `x` is calculated as follows:

$$x \text{ position of each hierarchy} = (\text{file index} \times \text{space taken by each hierarchy}) + 20 \quad (5.2)$$

At the start of each drawing we need to initialise some attributes:

- The node size; width and height of a node, the default is one but if zoom slider value is changed then the node size takes the value of the new slider value position accordingly.
- The total depth level of the hierarchy which is calculated for each hierarchy using the `getDepth()` method.
- The `x` and `y` positions where the hierarchy should get plotted; where `x` and `y` are stored in the `listOfGraphPoints` list and `y` is always initialized with a constant value.
- The display panel height; height of the white panel where hierarchies are getting displayed.



- The hierarchy size, where nodes of each hierarchy can be found in the `arrayOfGraphs` list.
- The pixel size variable which is explained in the following paragraph

In order to fit thousands of nodes constructing the hierarchy onto the display screen, we need to overlap nodes. Accordingly, we need to know how many nodes are getting displayed proportional to the display screen to calculate the amount of overlapping space each node will occupy. The pixel size variable, `pixelSize` is then initialized depending on the number of nodes to be displayed representing a hierarchy and on the panel height where the hierarchy is observed, as represented in equation 5.3.

$$pixel\ size = \frac{panel\ height}{number\ of\ nodes\ to\ be\ displayed} \quad (5.3)$$

This value is then used to calculate the new y position of current node,  $y=y+pixelSize$ . This equation is used only when the number of nodes to be displayed exceed the display panel height. Otherwise, if the number of nodes are less than or equal to the display panel height, the nodes are displayed with a pixel size, meaning a width and height equal to 1 with no overlapping.

For a better visualization of the hierarchies placed against each other, the background colour of each hierarchy is alternated from white to gray and vice versa. When the hierarchy displayed is at an even position it has a greyed background. Same concept when displaying the bar charts visualizing the changes made between successive hierarchies, but instead of changing backgrounds we find it sufficient to only separate the charts with gray vertical lines.

All stated above explanation are pre-processing to the actual drawing on the display panel. A detailed explanation on how the indented plots are drawn on the display panel and how the variables are calculated and changed is presented below.

### **Hierarchy Plotting**

Parent-child relationships are expressed using indentation - omitting the edges - of the rectangular shapes representing the nodes. Each node is indented with respect to its depth level. To start with, we get the node's depth level. Accordingly we calculate the node's position on screen; nodes with higher depth value, we expect to find on the right. A node is positioned with respect to the hierarchy's `xorigin` position (the x position where the first node gets displayed) adding to it the node level decremented by 1 and multiplied by the node width, provided in equation (5.4). The y coordinate position

for each node displayed changes depending on the `pixelSize` value explained previously. The indented plot has a red to green colour coded gradient, which means, the `root` node is represented in `red` colour and colour degrades until reaching a `green` colour representing the leaf nodes and all intermediate nodes are represented in variant colour between both. The colour accompanied with each node is then dependent on the node depth, so the `red` component value degrades and the `green` component increments proportional with the node level. Each node is given its colour gradient as shown in equations (5.5) and (5.6).

$$x\ position = xorigin + (node\ level) - 1 \times node\ width \quad (5.4)$$

$$red\ colour\ component = 255 - (node\ level - 1) \times (255 \div (noOflevels - 1)) \quad (5.5)$$

$$green\ colour\ component = (node\ level - 1) \times (255 \div (noOflevels - 1)) \quad (5.6)$$

After drawing all nodes constructing the indented plot modelling a hierarchy, we store some points corresponding to the bound of each hierarchy for further processing. This is achieved by using the data structure `GraphBounds` (needed when user wants to select specific hierarchies). Each hierarchy has its nodes stored in a list. Lists for each hierarchy drawn is then stored in a container having all hierarchy lists.

At the top of each indented plot, the name of the file is placed.

```

H:Hierarchy representing a file and consisted of nodes (List <Node>)
n: node
xorig: x position where plotting starts (integer)
y: current vertical position
x: current horizontal position
leveln: current level(depth) of hierarchy (integer)
pixelSize: the width of the node(integer)
r: red component value(integer)
gc: green component value(integer)
while Node n: H do
    x=xorig+(leveln-1)*pixelSize;
    r=255-(leveln-1)*(255/(noOflevels-1));
    gc=(leveln-1)*(255/(noOflevels-1));
    rect=new Rectangle2D.Double(x, y, width, height);
    DrawRect(rect,n);
    //to store the position of each rectangle drawn with the corresponding
    node draw rectangle(x, y, width,height);
    y:=y+pixelSize;
end

```

**Algorithm 3:** Hierarchy plotting algorithm

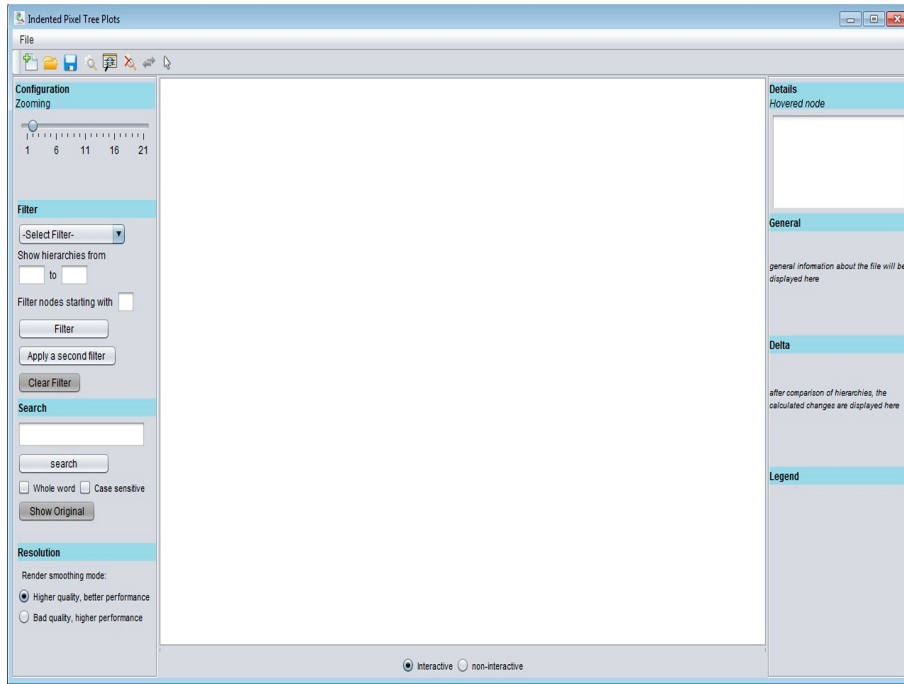
### Line Plotting

A description on how comparison lines are plotted to visualize the changes between hierarchies. In order to display lines visualizing the comparison applied on all hierarchies or specific hierarchies we use `Line2D` and `Path2D` Objects. To indicate if a comparison took place or not, to see if the lines will get drawn, `boolean` variables like `compareAll` and `compare` are used and are set to `true` whenever comparison is needed.

## 5.6 IPTPtool

This is the class implementing the Graphical User Interface. The tool is composed of a horizontal tool bar situated at the top of the mainframe and four main panel. A left and a right vertical panels, a central panel where hierarchy visualization will be observed and a bottom panel for display mode configuration and an additional panel is displayed under the central panel after a hierarchy comparison.

The right and left panels each is subdivided into smaller panels, where each sub-panel



**Figure 5.4:** The Graphical User Interface of the Tool

acts as a container for components such as *Combo Box*, *Slider*, *Radio buttons*, *Text fields*, *Labels* and *Text area*, where each component can be used as follows:

- **ComboBox:** to select a filter to apply on the hierarchies displayed
- **Slider:** for zooming in/out on hierarchies
- **Radio Buttons:** to select between modes like high quality, low quality, interactive or non-interactive
- **Text fields:** used for searching on specific nodes by entering part or whole desired node name, and can also be used in filtering
- **Labels:** to display titles for the sub-panels and represent results of the hierarchies compared
- **Text area:** To display details of a hovered node, like its name, parent's name, depth, index and number of children

# Chapter 6

## Conclusion

In this thesis, we have introduced an interactive visualization tool that supports the analysis of large dynamic hierarchies using IPTPs. We used the IPTPs technique to efficiently explore and compare large hierarchies because these plots have a high degree of visual scalability. The tool is configured to work with hierarchies in a newick file format. It provides a number of interactive features that help analyzing and distinguishing the changes between different IPTPs in different timesteps. These features are zooming in/out displayed IPTPs, collapse/expand nodes, subhierarchy selection for a larger display, searching a specific node, details on demand by hovering over nodes and lines and displaying detailed information, applying filters such as displaying only leaf nodes or show only removed nodes. Users can also choose between displaying a high quality mode or a low quality mode for a faster navigation since rendering high quality takes more time. The user can as well choose to disable the interaction of hierarchies and only display a screenshot.

Hierarchies are displayed aligned using a side-by-side representation of IPTPs. To achieve a mental map preserving overview-based diagram we have shown the dynamics of a hierarchy by a static representation and illustrated the changes between subsequent hierarchies by special links - straight ones and also curved ones serving as comparison lines.

We demonstrated how the interactive features can be applied to explore, compare, filter and highlight hierarchies on a number of different datasets. We applied it to a very large dataset and also to smaller ones in our illustrative case studies. The NCBI taxonomy, a hierarchical dataset containing 324,276 nodes classifying species and organisms, is an example of a huge dataset for which we proved the ability to examine it using subregion selection to have a closer view to a subhierarchy. Then smaller datasets containing

from 100 to 10,000 nodes on average were also tested, representing the evolving prefix tree structure of words occurring in paper titles from all papers in the field of computer science.

## Limitations

An improvement could be done to the newick file parser to accelerate the parsing of huge files. Also the subregion selection feature needs to be improved since the enlargement of the selected region can overlap the IPTP next to it. In addition to that, when the subhierarchy selection feature is used all other features and functionalities have no effect on the IPTPs displayed. A time filter uses the name of files for the filtering process, that means that this type of filter will not work if the file names were otherwise stated which means if they do not have the year stated at the end of the file name.

## Future Work

Interactivity on visualized hierarchies is a non-ending cycle, there are always filters that could be applied and different types of interactions to implement. For example, in our tool zooming in/out features can only be applied to all hierarchies represented on screen. Zooming in could be done on a subhierarchy not only on the whole hierarchies displayed. Also extended filters need to be implemented, like the ability to display a subhierarchy selected by the user and to apply it to all hierarchies displayed, putting into consideration that this subhierarchy chosen by the user could be changed in subsequent hierarchies.

The tool can also provide a better view on the bar charts representing the changes between two successive IPTPs, that could be done by clicking on the bar chart, a larger bar chart could be displayed on a pop-up panel with detailed information with the percentage of changes (Delta, explained in Section 6.4).

To visualize the changes of the dynamic hierarchies we use coloured lines as mentioned in this work. When a big number of lines overlap or these are really close to each other this can cause visual clutter. In order to reduce the visual clutter we can use edge bundling as mentioned in the work of Danny Holten [14].

# Appendix A

## User Manual

Our tool is a basic one consisting of a menu bar and a tool bar positioned at the top of the mainframe, and four main panels that are shown on the start up of the tool as shown in Figure ???. An additional panel is observed when a comparison between hierarchies takes place. The panels are described as follows:

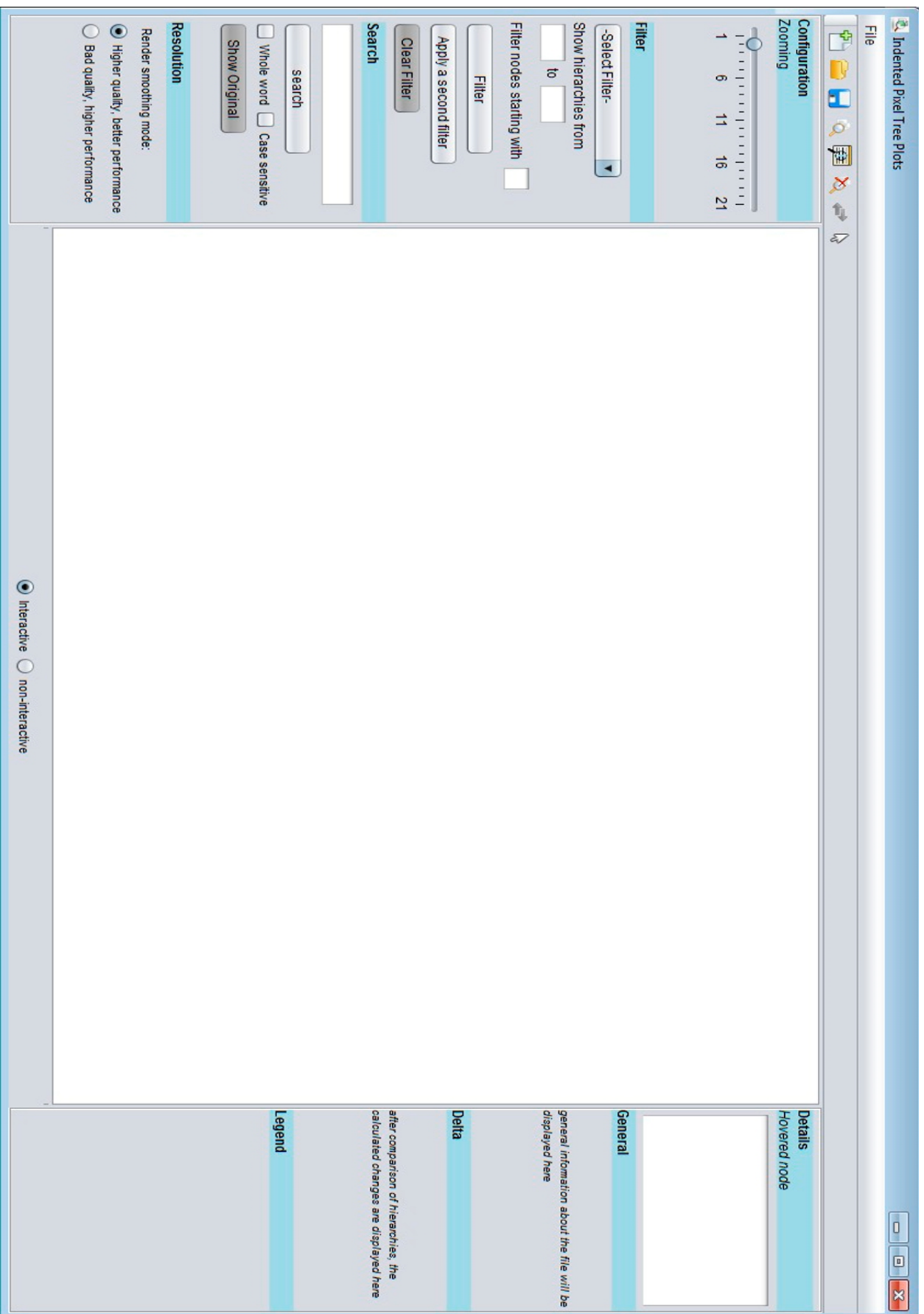
- Configuration panel: used for configuration purposes
- Detail panel: used to display detailed information about the hierarchies
- Control panel: composed of radio buttons for display mode purposes
- Hierarchy panel: where the hierarchies (IPTPs) will be visualized
- Bar chart panel: an additional panel that will be displayed at the bottom of the central panel to display some comparison statistics in a bar chart form, only when comparison between hierarchies takes place.

A description of all panel components and functionalities are provided next.

### A.1 Menu Bar

The menu bar is located at the top of the tool's mainframe. It includes menu items and options specific to the tool. Items located within the menu bar are also found in the tool bar. File menu is the menu item provided in our menu bar which consists of options such as New File, Open File, Save and Close.








Please refer to the next section for a detailed explanation.



**Figure A.1:** The graphical user interface of the Indented Pixel Tree Browser provides a variety of interactive features to manipulate the hierarchical data, to annotate and highlight subregions, to compare trees, and to navigate in it.



## A.2 Tool Bar

The tool bar is located at the top of the mainframe as shown in Figure A.2, directly under the **Menu bar**. It serves as an always-available, easy-to-use interface for performing common functions. It consists of a set of button elements; 'open folder' button , 'save' button , 'compare'  and 'compare all'  buttons, 'clear comparison' button , 'delta' button  and 'default cursor' button  described as follows:

- new file: to create a new file
- open folder: to open a directory
- save: to save IPTPs displayed on the **Hierarchy** panel (display panel)
- compare: to compare between selected hierarchies
- compare all: to compare all displayed hierarchies
- clear comparison: clears all comparison lines drawn between the IPTPs
- delta: displays the changes in percentage between two consecutive IPTPs on the **Detail** panel
- default cursor: restore the cursor to its default

### New File

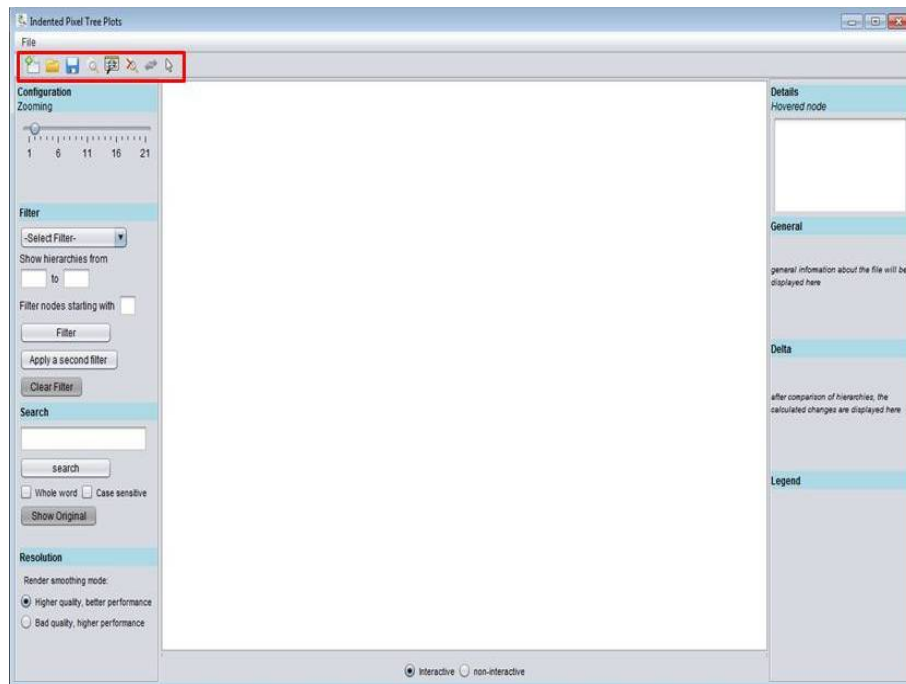
After the new file button situated at the top left corner of the toolbar is pressed, a new blank display panel is created for a new file to get displayed.

### Open Folder

By pressing the open folder button, a file chooser window pops out and a directory having files to be displayed can be chosen and displayed on screen.

### Save

Pressing the save button saves the IPTPs viewed on the **Hierarchy** panel in a JPEG image format in the directory chosen by the user.



**Figure A.2:** Tool bar

## Compare/Compare All


Comparison between hierarchies can be done in two different ways, either by selecting the hierarchies to compare or by comparing all hierarchies displayed.

By pressing the compare button, only selected hierarchies will be compared, and by pressing the compare all button, all displayed hierarchies will be compared. After pressing any of the two buttons, coloured lines are observed which describe changes between hierarchies. Red coloured lines demonstrate removed nodes, green coloured lines demonstrate added nodes and blue coloured lines demonstrate switched position nodes. A description of the line, like the **type** (added, removed or switched) and the **name** of the node are observed when hovering on any of the lines displayed between hierarchies. For example when a blue line is hovered, the type displayed is "switched" and the name corresponding to the node is displayed next to the type.

## Clear


After pressing the clear button, comparison lines - if any - will be removed.

## Delta

A comparison needs to take place before pressing the delta button. When the 'delta' button is pressed, the cursor icon changes to  and the user can put the cursor between any consecutive IPTPs to display the changes in percentage of: the added nodes, removed

nodes and switched position nodes between the first hierarchy (the one on the left of the cursor) and the second hierarchy (the one on the right of the cursor). The calculated percentages can be observed on the "Delta" section of the Detail panel, positioned at the right of the tool's mainframe.

### **Default cursor**

To restore the default cursor when the cursor is not in its default state, the user needs to press the default cursor button . This is mainly applied when the delta button is used.

## **A.3 Configuration Panel**

This panel appears where any configuration or alternation on hierarchies displayed takes place. The panel consists of four parts: Zoom, Filter, Search and Resolution, where each part can consist of more than one component:

- Zoom: consists of a slider bar for zooming purposes
- Filter: to apply certain filters on the IPTPs displayed
- Search: to search for a certain node on the display panel or filter nodes starting with certain character
- Resolution: to select between high quality or low quality display

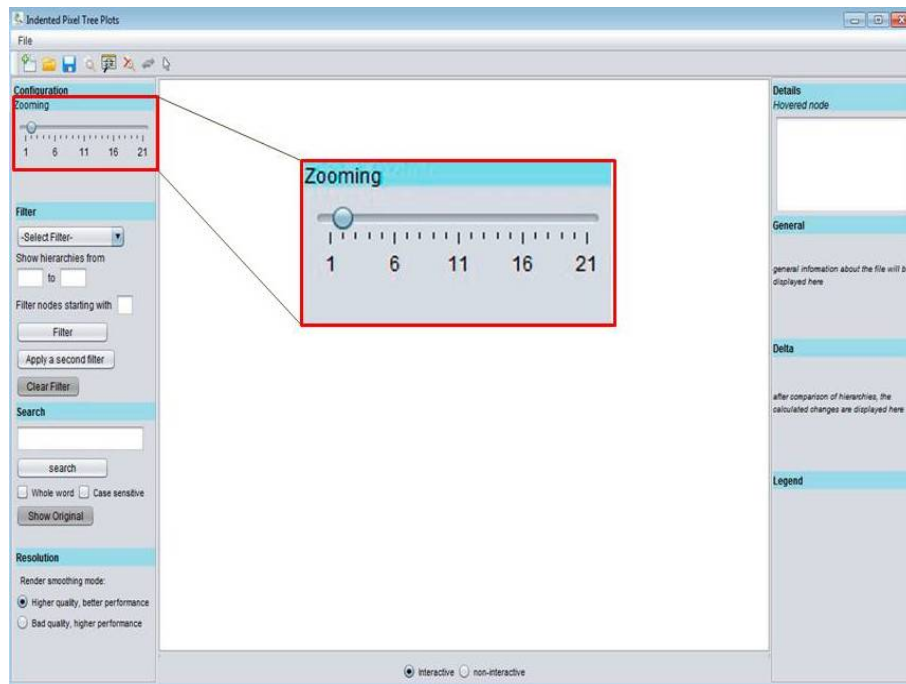
### **Zoom**

A zoom in/out of the hierarchies displayed can be observed on hierarchies when the mouse is pressed and moved on the slider bar. The hierarchies get larger or smaller according to the value on the bar the mouse stopped at.

### **Filter**

There are three ways to apply filters on the IPTPs displayed, either by selecting a filter from a drop down list having some defined filters, or using a time filter or an alphabet filter as seen in Figure A.4.

- Drop down list: has 4 defined filters, "show only removed nodes", "show only added nodes", "show only changed position nodes" and "show only leaf nodes". First three filters are only used when comparison takes place.



**Figure A.3:** Zoom slider

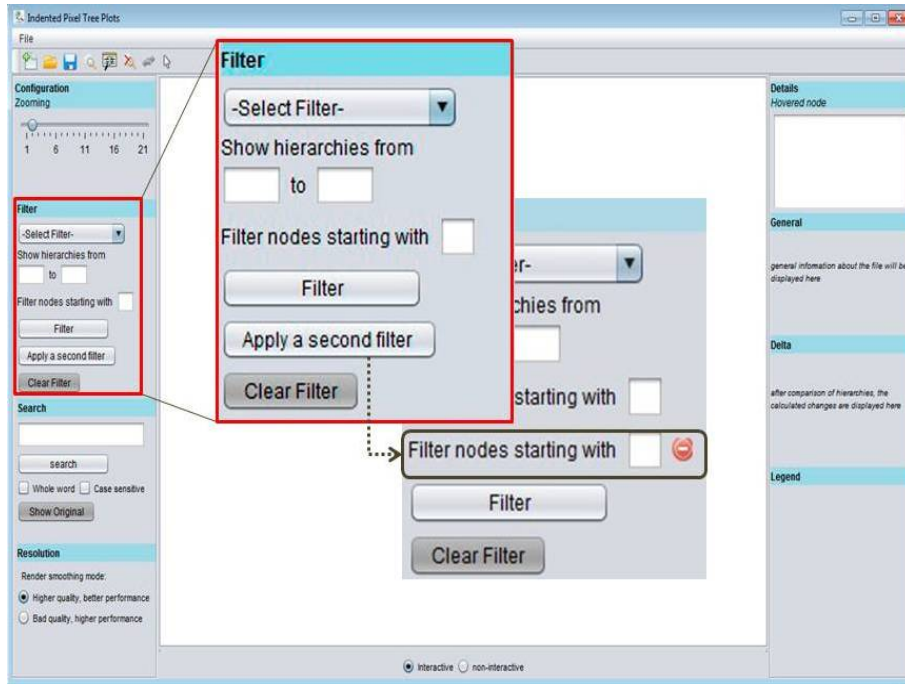
- Time filter: to display only desired IPTPs depending on a year range. The 'start date' (in years) can be displayed in the first text field and the 'end date' in the second text field
- Alphabet filter: to display only nodes that start with a desired alphabet (A1). An additional alphabet filter (A2) can be applied by pressing the 'Apply a second filter' button to give the user the ability to filter out attributes starting with two different characters at the same time.

These filters are applied on the hierarchies displayed after the 'Filter' button is pressed except when choosing from the drop down list. And accordingly, the nodes to be filtered are shown and displayed in their colour code according to their depth level, and all other nodes are filtered out and displayed in **grey** colour.

All three types of filters can be applied at the same time. In order to clear the filters and display the original IPTPs, a 'Clear Filter' button needs to be pressed.

## Searching

Searching a specific node only requires a String to be entered in the text field shown in Figure A.5 followed by a click on the "Search" button. Nodes searched for are displayed in their colour code according to their depth level and all other nodes are filtered out by displaying them in grey colour. If a filter is applied and the nodes resulting from the



**Figure A.4:** Filter

search button are already shown by another filter, then the node is coloured in yellow to distinguish the search result from the filter result as shown by the blue rectangle in Figure A.5. Options like 'whole word' and 'case sensitive' can be applied to the searching process by clicking on either, or both check boxes associated with each option.

A 'Show original' button can be used to clear the search text field and remove the coloured nodes displayed resulting from the search previously done.

## Resolution

Alternation between high quality and low quality is done using radio buttons. When faster processing is needed then choosing low quality is preferable.

## A.4 Detail Panel

All the details on IPTPs displayed can be found on the right side of the tool's mainframe. This panel is composed of four parts displaying details about something selectable:

- Hovered nodes: displays information of a certain node
- General information: for general file information
- Delta: displays numbers explaining how much a hierarchy has changed from its predecessor

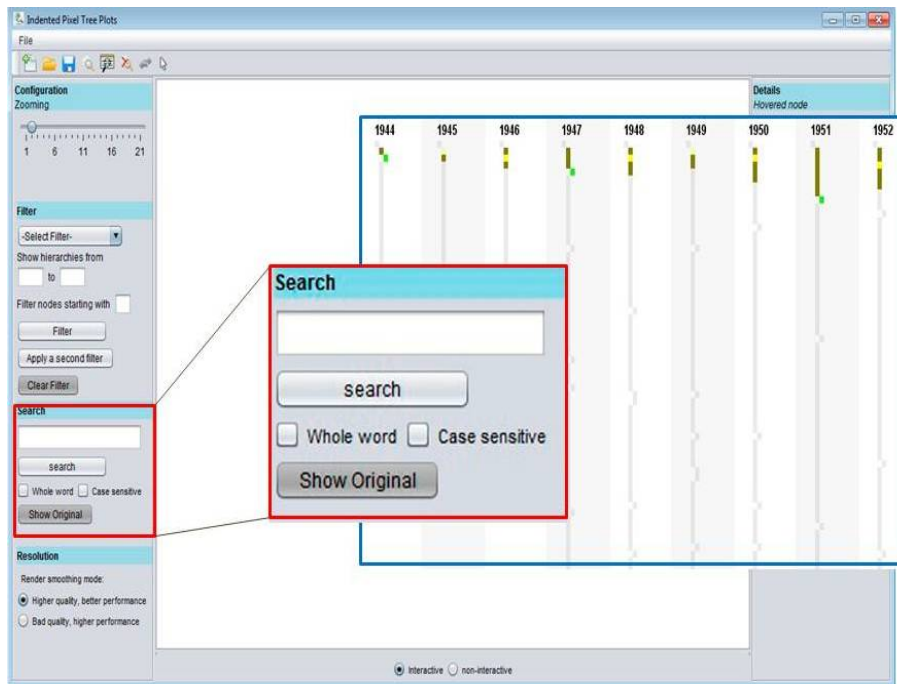


Figure A.5: Search engine

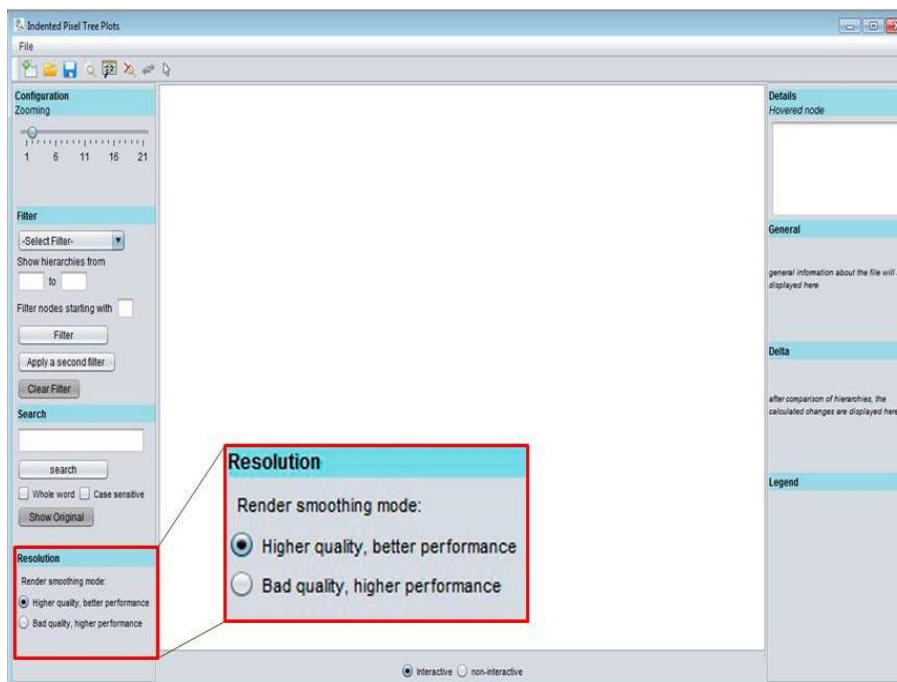
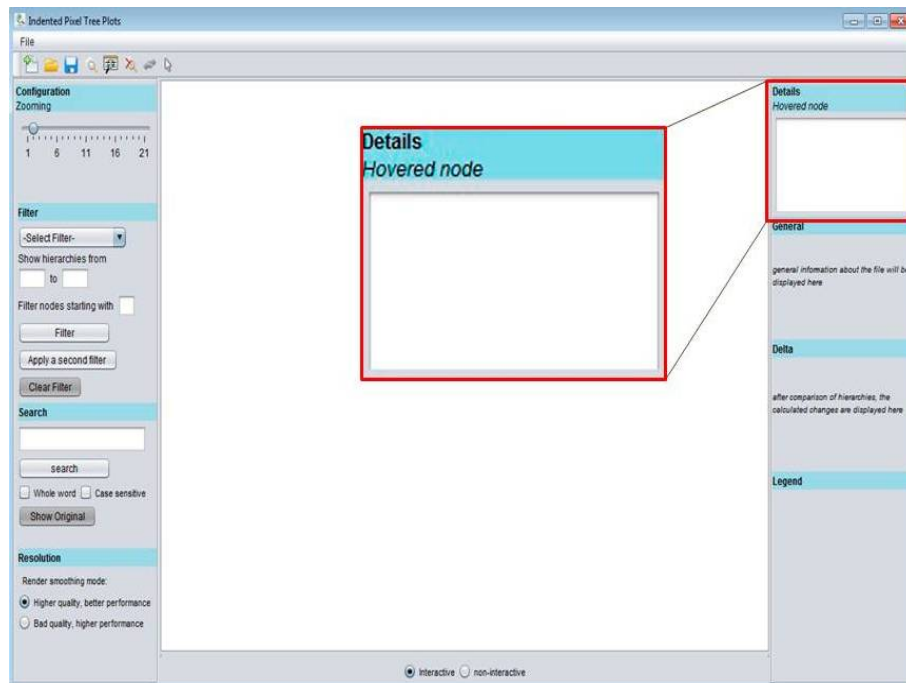


Figure A.6: Resolution mode

- Graph legend: describes the meaning of the colors used for the visualization

## Hovered Nodes

When the mouse cursor is located over a node displayed, all information about that specific node like the node name, its parent name, its depth level, its index (its position in the hierarchy) and its number of children are displayed in the top right corner of the mainframe in the text area provided as shown in Figure A.7.



**Figure A.7:** Hovered nodes details area

## General

After hierarchies are displayed, the information of the first hierarchy displayed, like the file name and the maximum depth of the hierarchy can be shown. When the mouse is pressed on other IPTPs representing different files, data displayed changes accordingly.

## Delta

After a comparison takes place, changes are calculated and the percentage of change (added, removed, changed position nodes) between the first IPTP and the last IPTP displayed can be observed on the panel as area shown in Figure A.9. This is also where the changes can be observed when choosing to calculate the delta using the "delta" button explained previously in the "Tool bar" section. The values displayed are as follows:

- Added nodes: represents the percentage of added nodes
- Removed nodes: represents the percentage of deleted nodes

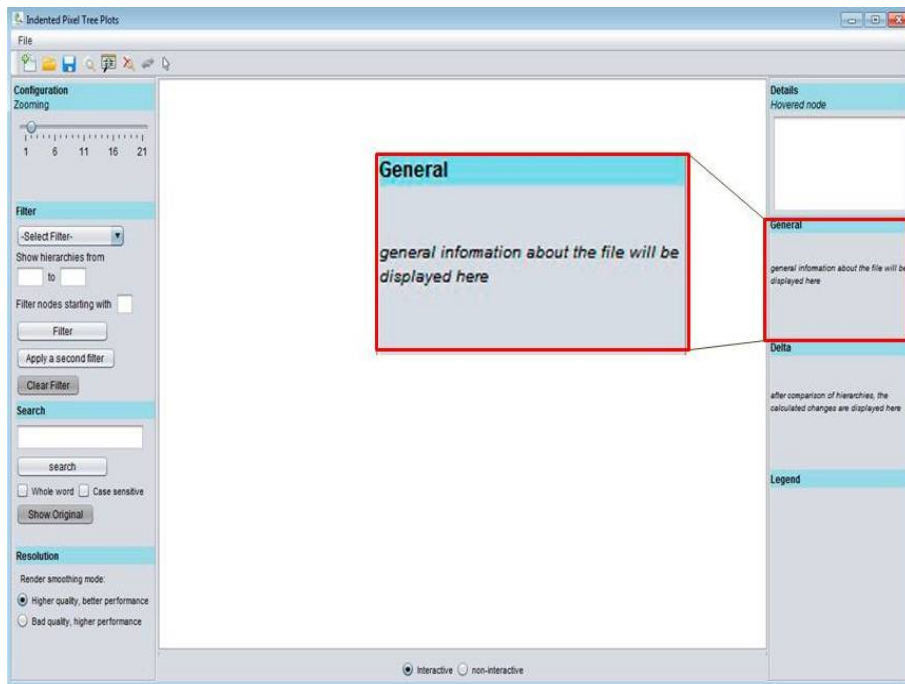


Figure A.8: General file details

- Changed position nodes: represents the percentage of changed position nodes

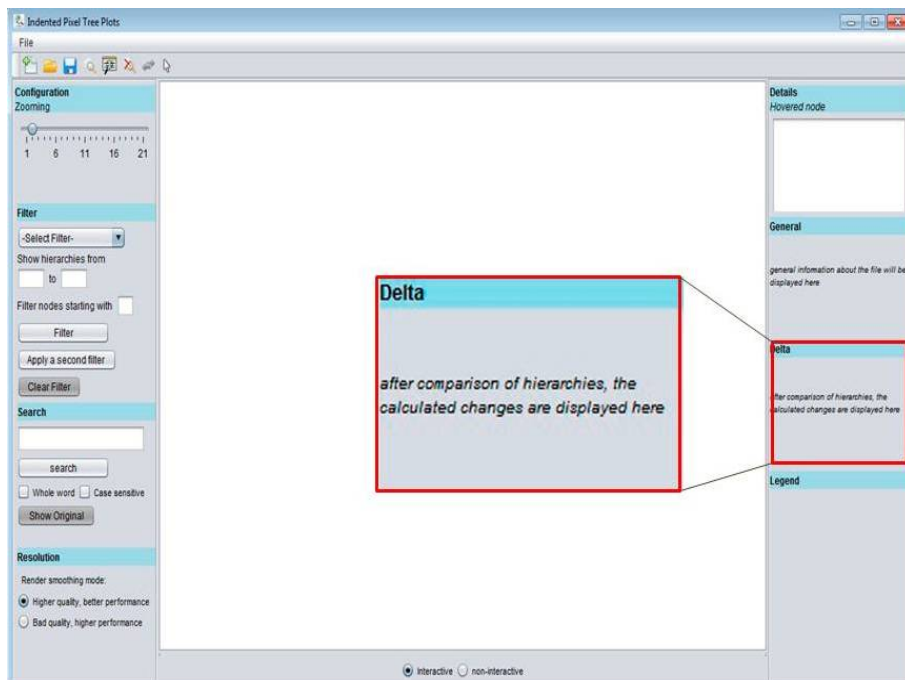


Figure A.9: Delta



## Graph Legend

A graph legend is used to describe the meaning of colors used to visualize the IPTPs and their changes. A horizontal colour stripe is used to represent the colour codes (according to the node's depth level) of the IPTP. When a comparison takes place a 'comparison line' legend and a 'bar chart' legend are displayed as shown in Figure A.10.

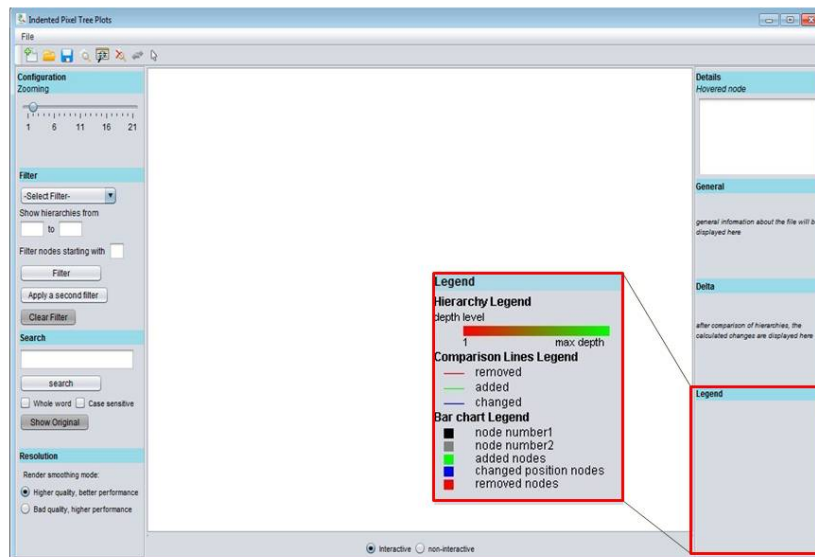


Figure A.10: Graph legend

## A.5 Control Panel

In order to make the display panel (positioned at the center of the mainframe) interactive or non-interactive, two radio buttons are displayed at the bottom for alternation between modes. The non-interactive mode only allows the user to hover over nodes and scroll horizontally or vertically. It is a saved image of what was displayed right before changing from interactive to non-interactive mode. This can be used when interaction with the displayed hierarchies is not needed, to make processing faster when moving around all hierarchies when an observation is needed.

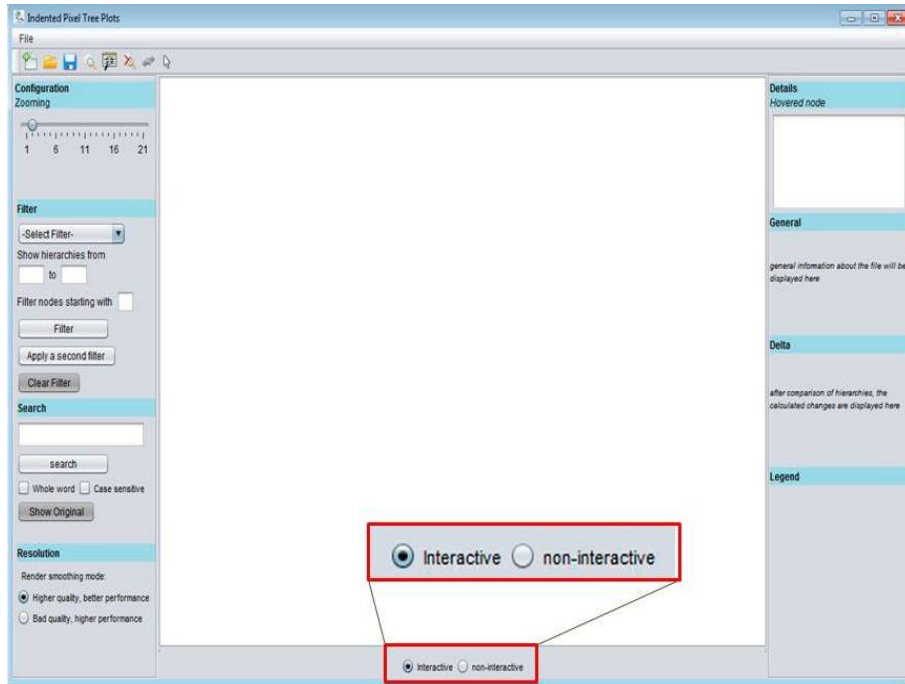


Figure A.11: Control modes

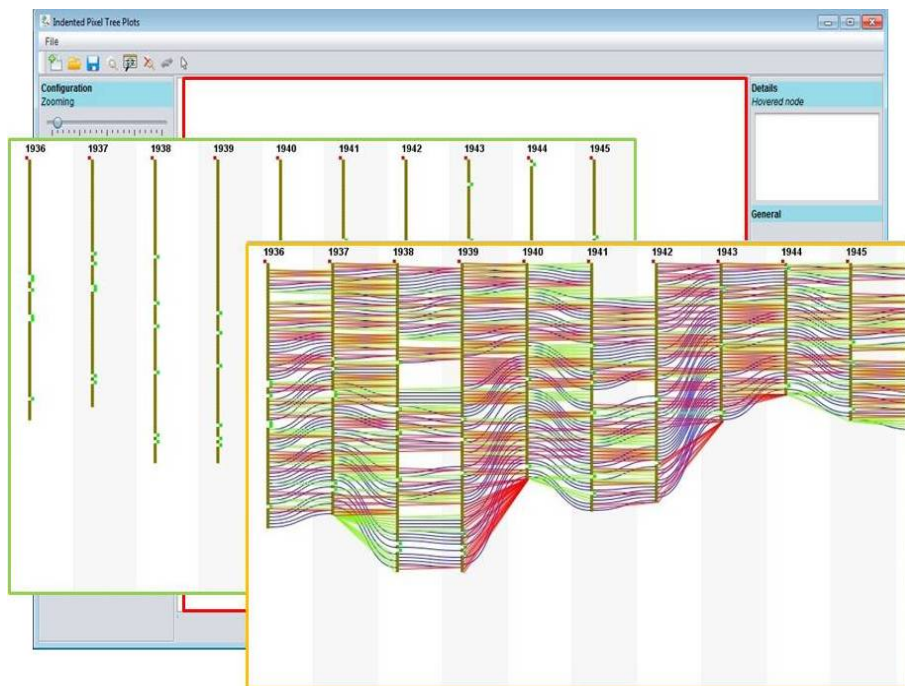


Figure A.12: IPTPs display area

## A.6 Hierarchy Panel

The Indented Pixel Tree Plot (IPTP) is visually depicted at the center of the mainframe surrounded by a red rectangle in Figure A.12. The green rectangle represents the display

area after the user selection on a specific directory, as for the orange rectangle, it illustrates the display area after the compare all button is pressed. Any alternation on the IPTPs displayed like filtering affects the main display.

## A.7 Bar Chart Panel

After a comparison takes place (any of the two buttons, compare/compare all are pressed), the changes between all hierarchies displayed is visualized using bar charts, one for each two consecutive hierarchies. The bar chart consists of five bars, 2 for representing the number of nodes in each file and the other 3 representing the changes between hierarchies (added, removed and switched nodes). The size of the first hierarchy is represented in black colour, the size of the second hierarchy is represented in grey colour, the number of added, removed and changed place nodes are each represented in green, red and blue colors respectively. The red rectangle in Figure A.13 represents the bar chart panel after comparison.

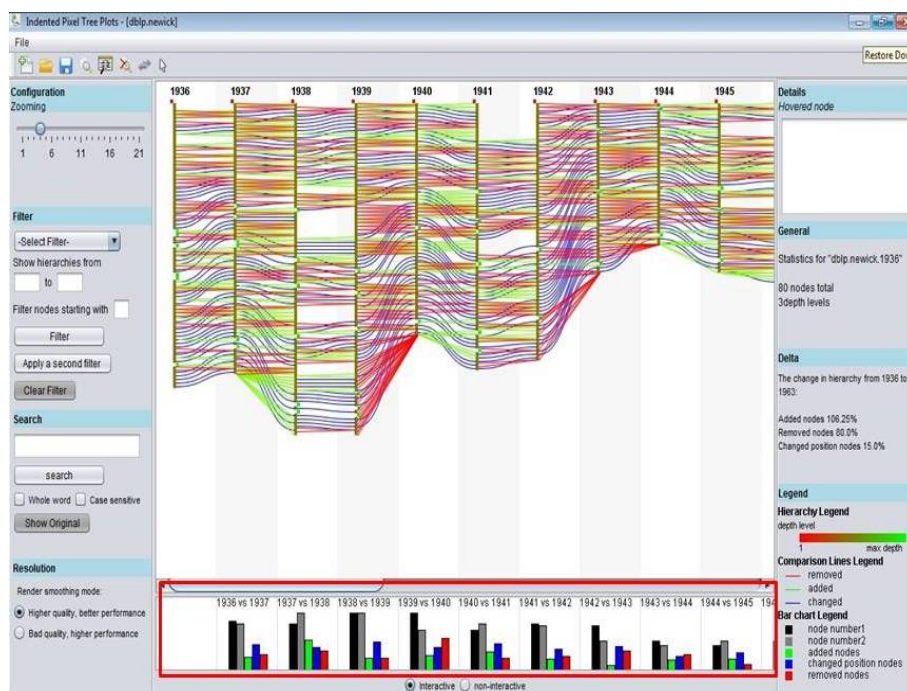


Figure A.13: Bar chart



# Bibliography

- [1] K. Andrews and H. Heidegger. Information slices: Visualising and exploring large hierarchies using cascading, semi-circular discs. In *Proc of IEEE Infovis 98 late breaking Hot Topics*, pages 9–11, 1998.
- [2] K. Andrews and J. Kasanicka. A comparative study of four hierarchy browsers using the hierarchical visualisation testing environment (hvte). In *Information Visualization, 2007. IV'07. 11th International Conference*, pages 81–86. IEEE, 2007.
- [3] M. Balzer and O. Deussen. Voronoi treemaps. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 49–56. Ieee, 2005.
- [4] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, pages 165–172. ACM, 2005.
- [5] G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.
- [6] B.B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *AcM Transactions on Graphics (TOG)*, 21(4):833–854, 2002.
- [7] M. Bruls, K. Huizing, and J.J. Van Wijk. Squarified treemaps. In *Proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. Citeseer, 2000.
- [8] M. Burch, M. Raschke, and D. Weiskopf. Indented pixel tree plots. *Advances in Visual Computing*, pages 338–349, 2010.
- [9] M. Burch, H. Schmauder, and D. Weiskopf. Indented pixel tree browser for exploring huge hierarchies. *Advances in Visual Computing*, pages 301–312, 2011.

- [10] S. Diehl, C. Görg, and A. Kerren. Preserving the mental map using foresighted layout. In *Proceedings of Joint Eurographics–IEEE TCVG Symposium on Visualization (VisSym01)*, pages 175–184, 2001.
- [11] P. Eades and Fujitsu Laboratories. International Institute for Advanced Study of Social Information Science. *Preserving the mental map of a diagram*. International Institute for Advanced Study of Social Information Science, Fujitsu Limited, 1991.
- [12] J. Heer, M. Bostock, and V. Ogievetsky. A tour through the visualization zoo. *Communications of the ACM*, 53(6):59–67, 2010.
- [13] I. Herman, G. Melançon, and M.S. Marshall. Graph visualization and navigation in information visualization: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):24–43, 2000.
- [14] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):741–748, 2006.
- [15] P. Irani, D. Slonowsky, and P. Shajahan. Human perception of structure in shaded space-filling visualizations. *Information Visualization*, 5(1):47, 2006.
- [16] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization’91, Proceedings., IEEE Conference on*, pages 284–291. IEEE, 1991.
- [17] D.E. Knuth. The art of computer programming, volume i: Fundamental algorithms, chapter 2, 1973.
- [18] J.B. Kruskal and J.M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.
- [19] M.J. McGuffin and J.M. Robert. Quantifying the space-efficiency of 2d graphical representations of trees. *Information Visualization*, 9(2):115–140, 2010.
- [20] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of visual languages and computing*, 6(2):183–210, 1995.
- [21] Q.V. Nguyen and M.L. Huang. A space-optimized tree visualization. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 85–92. IEEE, 2002.
- [22] B. Nouanesengsy and Y. Li. Hierarchy visualizations. In *IN M. SHAH CHARACTERS. PROCEEDINGS OF THE 27 & R. JAIN (EDS.), MOTION-BASED RECOGNITION*. Citeseer, 1997.

- [23] E.M. Reingold and J.S. Tilford. Tidier drawings of trees. *Software Engineering, IEEE Transactions on*, (2):223–228, 1981.
- [24] E.W. Sayers, T. Barrett, D.A. Benson, E. Bolton, S.H. Bryant, K. Canese, V. Chetvernin, D.M. Church, M. DiCuccio, S. Federhen, et al. Database resources of the national center for biotechnology information. *Nucleic acids research*, 39(suppl 1):D38–D51, 2011.
- [25] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on graphics (TOG)*, 11(1):92–99, 1992.
- [26] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.
- [27] B. Shneiderman. Treemaps for space-constrained visualization of hierarchies. *ACM Transactions on Graphics (TOG) Volume*, 11:92–99, 1998.
- [28] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on*, pages 73–78. Ieee, 2001.
- [29] J. Stasko, R. Catrambone, M. Guzdial, and K. McDonald. An evaluation of space-filling information visualizations for depicting hierarchical structures. *International Journal of Human-Computer Studies*, 53(5):663–694, 2000.
- [30] J. Stasko and E. Zhang. Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 57–65. IEEE, 2000.
- [31] A. Telea and D. Auber. Code flows: Visualizing structural evolution of source code. In *Computer Graphics Forum*, volume 27, pages 831–838. Wiley Online Library, 2008.
- [32] F. Van Ham and J.J. van Wijk. Beamtrees: Compact visualization of large hierarchies. *Information Visualization*, 2(1):31–39, 2003.
- [33] J.J. Van Wijk and H. Van De Wetering. Cushion treemaps: Visualization of hierarchical information. In *Information Visualization, 1999.(Info Vis' 99) Proceedings. 1999 IEEE Symposium on*, pages 73–78. IEEE, 1999.
- [34] F. Vernier and L. Nigay. Modifiable treemaps containing variable-shaped units. In *IEEE information Visualization*, 2000.

- [35] A. Vilanova, A. Telea, G. Scheuermann, and T. Möller. Visual comparison of hierarchically organized data.
- [36] Kai Wetzal. Pebbles - using circular treemaps to visualize disk usage. <http://lip.sourceforge.net/ctreemap.html>, 2003. Online; accessed 15-September-2012.
- [37] J. Yang, M.O. Ward, E.A. Rundensteiner, and A. Patro. Interring: a visual interface for navigating and manipulating hierarchies. *Information Visualization*, 2(1):16–30, 2003.
- [38] S. Zhao, M.J. McGuffin, and M.H. Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 57–64. IEEE, 2005.



## **Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben.

Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.

Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.

Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.

Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

(Christine Louka)