BarcodeTree: Scalable Comparison of Multiple Hierarchies

Guozheng Li, Yu Zhang, Yu Dong, Jie Liang, Jinson Zhang, Jinsong Wang,
Michael J. McGuffin, and Xiaoru Yuan

Abstract— We propose BarcodeTree (BCT), a novel visualization technique for comparing topological structures and node attribute values of multiple trees. BCT can provide an overview of one hundred shallow and stable trees simultaneously, without aggregating individual nodes. Each BCT is shown within a single row using a style similar to a barcode, allowing trees to be stacked vertically with matching nodes aligned horizontally to ease comparison and maintain space efficiency. We design several visual cues and interactive techniques to help users understand the topological structure and compare trees. In an experiment comparing two variants of BCT with icicle plots, the results suggest that BCTs make it easier to visually compare trees by reducing the vertical distance between different trees. We also present two case studies involving a dataset of hundreds of trees to demonstrate BCT's utility.

Index Terms—tree visualization, comparison, multiple trees

✦

1 INTRODUCTION

Datasets sometimes comprise of many similar trees (such as different collections of books at different local libraries, all classified according to the Dewey Decimal system) or a single tree that changes over time (e.g., the "site map" of a website changing over time). Visual overviews of such collections of trees could greatly help with performing comparisons, identifying nodes or subtrees that are present or absent, or finding trends and outliers.

There is much previous literature [54] on visualizing individual trees, and considerably less on visual comparison of trees, mostly limited to visualizing two trees at a time [5, 15, 29, 30, 34, 44, 63]. The previous work that has demonstrated the visualizations of three or more trees simultaneously has shown less than 40 trees *at most* [9]. As sources of all kinds of data continue to grow, there is an increasing need to visualize larger numbers of multiple trees.

We propose BarcodeTree (BCT) for scalable comparison of *both* topological structure and node attribute values across multiple trees. BCT linearizes each tree as a single row (Fig. 1), compressing the visualization vertically, making it easy to stack multiple trees and align their matching nodes (Fig. 2), reducing the distance between matching nodes. We investigate two variants, BCT_w (width-encoded BarcodeTree) and BCT_h (height-encoded BarcodeTree), which encode node depth into the width or height, respectively, of rectangles. Node attributes can be encoded with color, or (in the case of BCT_w) in the height of the rectangles. To help users understand the topological structure, we also design "structural cues" to highlight the descendants, ancestors, and siblings of the node under the cursor.

The design of BCTs was motivated by working with datasets of trees that are relatively shallow and *stable*. By "stable", we mean that there is a pre-defined universal tree which is a supertree of all the trees in the dataset. The differences between the trees are that they may contain different subsets of the nodes in the supertree and that the attribute
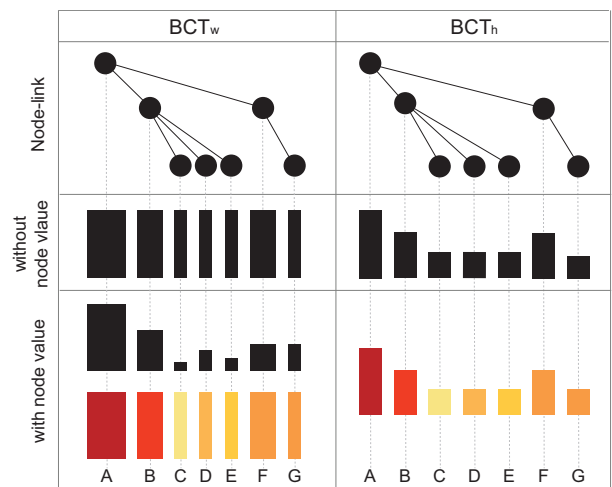
- *Guozheng Li and Xiaoru Yuan are with Key Laboratory of Machine Perception (Ministry of Education) and the National Engineering Laboratory for Big Data Analysis and Application, Peking University. E-mail: {guozheng.li, xiaoru.yuan}@pku.edu.cn. Xiaoru Yuan is the corresponding author.*
- *Yu Zhang is with University of Oxford, United Kingdom and worked on this project during his undergraduate in Peking University.*
- *Yu Dong, Jie Liang, and Jinson Zhang are with University of Technology Sydney, Australia.*
- *Jinsong Wang is with Southwest Electric and Telecom Engineering Institute.*
- *Michael J. McGuffin is with Ecole de technologie superieure, Canada.*

Fig. 1. BarcodeTrees (BCTs) linearize hierarchical data and map the nodes to rectangles. The BCT_w encodes the depth of each node with the *Width* of each rectangle, leaving height and color available for encoding node attributes. The BCT_h encodes depth with the *Height* of the rectangles, leaving color available for encoding node attribute values.
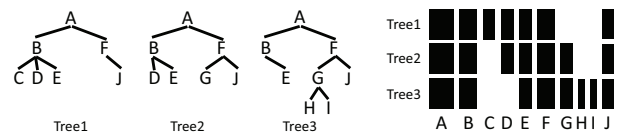

Fig. 2. A sequence of three trees, visualized using classical node-link drawings (left) and then stacked as three BCT_w rows. By aligning matching nodes, the differences in the structures become apparent.

values on each node may change. Such datasets of trees are encountered in applications such as (1) daily borrowing records from a single university library for multiple days where books follow a pre-defined classification system; (2) collections of books in multiple libraries; (3) the budgets for multiple departments from different governments or the same government over time; (4) the categorization hierarchy of products on an e-commerce website that grows over time.

We compared our two variants of BCT with icicle plots [40] and Indented Pixel Tree Plots (IPTP) [13] in a pilot study, and found that time spent on IPTP was slower than icicle plots for all the tasks studied. A controlled experiment then compared BCT_w, BCT_h, and icicle plots, finding that BCTs require significantly more time in tasks requiring the understanding of the topology of an individual tree, but significantly less time in tasks involving the comparison of multiple trees.

Our contributions are: (1) BarcodeTree, a novel visualization technique for comparing the topological structure *and* node attribute values of multiple shallow and stable trees, capable of accommodating one hundred trees, at hundreds of nodes per tree (but with small fan-out degree), within a normal screen size, (2) visual feedback ("structural cues") to help users understand the topological structure, (3) a controlled experiment that demonstrates that BCTs can make tree comparison tasks significantly easier than with a competing visualization technique, and (4) two use cases involving a library book dataset to demonstrate the utility of BCTs.

2 RELATED WORK

2.1 Visualization of Individual Trees

Tree visualizations can be categorized into explicit and implicit techniques according to the visual encodings of parent-child relationships [54]. Explicit techniques encode these relationships with visible edges, e.g., line segments or arcs. Implicit techniques use inclusion (i.e., children nested, or enclosed, inside their parent) or adjacency [53].

Most *explicit* tree visualizations arrange hierarchical data within a 2D or 3D space, whereas arranging nodes in 1D is underexplored. Thread Arc [36] arranges the nodes of a tree along a 1D row according to chronology and uses circular arcs between nodes to encode parent-child relationships. However, the arcs used in Thread Arc make it difficult to compress the layout into a thin strip that would be amenable to stacking many trees together. In contrast, BCTs do not encode parent-child relationships with explicit visual elements, which increases space-efficiency for stacking multiple trees.

Within *implicit* tree visualizations, a prominent example of *inclusion* encoding is the treemap [35], which nests children inside their parents. In treemaps, leaf nodes show node attribute values through their size, and the attribute value of a non-leaf node is typically the sum of its children. By contrast, BCTs allow the attribute value of a node to be independent of the values of its children. Treemaps are usually arranged in 2D [10, 55, 57] or 3D [17, 60]. ArcTree [49] linearizes the hierarchical data and arranges nodes within a single row, which can be regarded as a "one dimensional" treemap. For example, a 7-node tree can be shown as an ArcTree with ▭▭▭▭ or without ▭▭▭ horizontal spaces between the borders of nodes. The same tree shown using BCT_h ▮▮▯▮ or BCT_w ▮▮▮▮▮ requires less spatial resolution, because BCTs contain fewer closely-spaced lines than ArcTrees.

Adjacency encodings, such as icicle plots [40], are a compromise between explicit and inclusion encodings. Compared to explicit encodings, the use of adjacency saves space by not showing explicit edges between nodes. Compared to inclusion encodings like treemaps, using adjacency is not as space-efficient [46], but makes the different levels of the tree easier to distinguish [4]. Icicle plots [40] show parent-child relationships with vertical adjacency: child nodes are placed below parent nodes. Each level of the tree is one row, with an entire tree occupying several rows. Although it is possible to stack multiple icicle plots vertically for comparison of trees, BCTs require much less vertical space than icicle plots. In this way, BCTs reduce the distance between the matching nodes of different trees.

BCTs can be understood as another kind of adjacency encoding. It maps the nodes of a tree to parallel bars in depth-first traversal order. Each parent node neither contains nor has explicit links to its children, but users can still discern the underlying topological structure. This is similar to the indented outline [37] layout (used in many file browsers) and to Indented Pixel Tree Plots (IPTP) [13], both of which position nodes according to depth-first traversal order. The indented outline layout works well for showing nodes with text labels. To enable users to read these text labels, indented outline always arranges nodes vertically rather than horizontally. If we remove the text labels from the nodes, compress and rotate 90 degrees, we obtain a layout similar to BCT_h, making it suitable for stacking multiple trees together. IPTP allocates a separate row for each level of a tree, like icicle plots, and therefore both of these layouts are less vertically compressed than BCTs.

2.2 Visual Comparison of Trees

Comparison is an essential task in data analysis [23, 24, 50]. A range of techniques exist for visually comparing trees [28], which we subdivide into: juxtaposition, merged views, and atomic representations.

Juxtaposition is the most prevalent strategy and can be done in space or time. *Spatial juxtaposition* can be done with node-link representations [9, 48, 69], radial representations [16, 25, 26], and treemaps [41]. Such side-by-side views can leverage interaction through brushing and linking and dynamic queries, highlighting the corresponding nodes of other trees interactively. However, without interaction, it is difficult to understand the highlighted differences of many nodes, making it ineffective for presenting the user with an overview of many changes across many trees. Furthermore, because the representations in previous work are not as vertically compressed as in BCTs (i.e., they are not "long and skinny"), these previous techniques do not scale well beyond a small number of trees. Previous works also juxtapose icicle plots for comparison. Some of these [15, 34, 44, 65] arrange two icicle plots adjacently and connect the matching leaf nodes with edges. By mirroring the icicle plots, Code flow [61] allows users to compare more than two trees. The explicitly drawn links provide an overview of topological structure difference, but also occupy much space, again preventing this technique from accommodating more than a small number of trees at a time. Taxonaut [68] juxtaposes multiple indented outlines horizontally and aligns them using a union tree, but it is still less scalable than BCTs because each indented outline needs to occupy several columns. Some works [18, 39, 43] connect matching nodes across trees using curved bands, but at the cost of "blurring together" individual nodes. *Temporal juxtaposition*, such as with TimeTree [14], uses animation to transition across different trees. This strategy also does not scale well to a large number of trees. Users have limited ability to remember variation between frames, and therefore have to view the animations multiple times.

Some previous works propose *merging* multiple trees into a single visual representation to improve space efficiency. Beck et al. [5, 6] propose a *dependency structure matrix* which arranges an icicle plot along each side of an adjacency matrix, with each cell in the matrix encoding the node differences. This approach only accommodates two trees at a time, and the matrix only shows the results of comparing leaf nodes. Some works [7, 20, 29, 30, 42, 63] merge the trees to be compared into one union tree. Graham et al. [27] explore several trees by merging them into a Directed Acyclic Graph (DAG) representation. Compared to a union tree, the DAG allows displaying multiple ancestor paths of a node. The merged view needs to show the differences among multiple trees. However, the relationships of multiple trees are complicated, especially when dealing with a large number of trees, and it is also difficult to show these results within a single merged view.

For sufficiently large numbers of trees, the limited screen space cannot show all the trees in full detail, so techniques have been developed that visualize the trees as *atomic representations*. Previous works [2, 33] visualize each tree as a single point in a scatter plot to support the comparison of large numbers of trees. The distances between two points show the degree of similarity between the associated trees. TreeEvo [22] mainly focuses on the structural heterogeneity of family trees. It visualizes each family tree with a pixel line and groups them into the nodes of a Sankey diagram. Explicit representation of relationships only reveals the comparison results from a high-level overview, but it cannot provide detailed comparisons of topological structure and node attribute values.

Despite the substantial previous works on tree comparison, to the best of our knowledge, previous methods are not able to show one hundred trees simultaneously in full detail, i.e., showing all individual nodes, showing both topological structure and node attribute values simultaneously on a typical PC screen. This is a unique feature of BCTs.

3 REQUIREMENTS ANALYSIS

We seek a technique for comparing multiple shallow and stable trees that is more scalable than previous techniques. We first consider previous literature on tree comparison to determine which tasks to support.

Guerra-Gómez et al. [29, 30] identified five types of tree comparison: (1) comparing topological differences where nodes have no attribute value; (2) comparing attribute values of leaf nodes with no changes in topology; (3) comparing attribute values of all nodes with no changes in topology; (4) comparing attribute values of leaf nodes with changes in topology; (5) comparing attribute values of all nodes with changes in topology. Munzner et al. [48] indicated that structural comparison is done by associating each node in one tree to its corresponding node in the other tree. In addition to comparison tasks, Chi et al. [16] also identified *global* and *local* tasks for multiple tree exploration. *Global* tasks refer to comparing multiple trees to discover trends/patterns. *Local* tasks refer to finding specific information about topology or node attributes. Supporting both global and local tasks agrees with Shneiderman's [56] recommendation to enable switching from an overview to a drilled-down view. We seek to support all the aforementioned kinds of tasks. Some additional tasks related to trees that could be considered are ways to edit trees (e.g., moving nodes or subtrees), but such editing tasks are beyond our scope since we seek to visualize the data.

Thus, our requirements for a new layout derive from three main needs. To support all of Guerra-Gómez et al.'s [29, 30] five tasks, we need a technique that shows both topology and node attributes, in a way that enables efficient comparison. To support *global* tasks, the technique should show as many trees as possible, while also showing as many individual nodes as possible within each tree. Providing a clear depiction of the topological structure of each tree is also desirable to support *local* tasks. However, making topology clear is less important than accommodating many trees and many nodes. This is because users can always start from an overview of many trees using the new layout, to compare multiple trees and discover trends or outliers, and later drill-down to one or a few trees shown with a more traditional layout that shows topology more clearly.

Below are our specific design requirements. R1 and R2 address the need for *global* tasks; R3 and R4 enable the five kinds of comparisons identified by Guerra-Gómez et al. and R5 enables *local* tasks.

R1: Show many trees within one screen. Providing an overview within the space of a typical display is essential for comparing multiple trees, and it is one of the requirements for many usage scenarios of tree comparison [9, 34, 61].

R2: Show many nodes within each tree. The visualization should show individual nodes within each tree. Providing such fine-grained information, while the user is still viewing an overview of the multiple tree dataset, reduces the need for the user to drill-down to more detailed views.

R3: Show one quantitative attribute for each node. Encode the attribute value on each node, to enable comparison of node values across trees, revealing trends and outliers.

R4: Facilitate comparison of a node's presence/absence or attribute values across many trees. It should be easy to find matching nodes across multiple trees, for comparing their attribute values, and for comparing the presence or absence of a node in different trees.

R5: Show topological (parent-child) relationships within each tree. It should be possible to discern the topological structure of tree data. This requirement is listed last to indicate that some sacrifices to the clarity of parent-child relationships are acceptable if such sacrifices increase space efficiency (**R1**, **R2**).

4 BARCODETREE DESIGN

Guided by the design requirements above, we propose BarcodeTree (BCT), which is a tree visualization to compare topological structure and node attribute values among multiple trees. In this section, we introduce the visual design of BCT, and then we propose interaction techniques to help users understand the underlying hierarchical data.

4.1 Visual Encoding Designs

To achieve scalability in the number of trees and nodes, we investigated mapping topology of multiple trees to vertical and horizontal positions (the most efficient visual channels [47]). Specifically, BCT visualizes each tree within a single row to minimize its vertical extent, which maximizes the number of trees that can be stacked vertically (**R1**). BCT

Fig. 3. "Structural cues" highlight certain nodes in response to the cursor hovering over a node, to make topological relationships more apparent. Horizontal underlining appears under ancestors to the left of the cursor, and also under descendants to the right of the cursor. Vertical tick marks appear under siblings at the same level as the node under the cursor. Both marks appear under the node under the cursor, resulting in a cross-shaped glyph. Above we see two variants of these structural cues, with an ellipse with dotted border indicating their differences: (a) Descendants are shown with a single horizontal stroke; (b) Each subtree is shown with a separate stroke.

also packs nodes horizontally within each row. Parent-child relationships are encoded into relative positions in a space-efficient manner, to maximize the number of nodes that can be packed horizontally (**R2**).

BCT represents each node as a rectangle, which can be rendered very efficiently, meaning that large numbers of nodes can be rendered at interactive frame rates (**R1**, **R2**). The visual channels of the rectangles (width, height, color) can encode the node depth and node attribute value (**R3**). Mapping the node depth to rectangle width produces BCT_w. Specifically, the root node has the largest width, and deeper nodes have smaller widths. The width difference between nodes at adjacent levels in BCT_w is identical. Mapping the node depth to rectangle height produces BCT_h, which makes it easier for a user to perceive the depth of each node. In the resulting visualization for a large number of trees, each node only occupies a tiny space to fit into the screen space. Specifically, the width and height of each node are small. The small length could encode the node depth because the target hierarchical data is relatively shallow. However, the node attribute value has a high dynamic range. Therefore the node attribute value can only be encoded into the node color, which is not limited by the node size. Encoding node attribute values into the color channel cannot support the accurate comparison. The remaining visual channel (height of BCT_w and width of BCT_h) can encode the attribute values repeatedly as a complementary in some cases. However, encoding the attribute value into the node width of BCT_h will lead to larger gaps between the nodes after the alignment and therefore, low space utilization efficiency. Therefore, we rule out this design.

BCTs utilize *pre-order* depth-first traversal to determine the horizontal node positions, which encode the parent-child relationships. Nodes in the hierarchy are added to the layout left-to-right according to the order of the first traversal time. The descendants of each node are placed to the right of the node. The user can find the descendants of a node N by scanning to the right of N until they encounter another node of the same width or height as N, depending on the variant of BCT used. Therefore, BCT allows users to interpret the tree's topological structure (**R5**). Because of its compactness, BCT can be embedded into a single line of text, just like sparklines [64]. For example, ▉▌▌▌▌▌▐ represents a tree with eleven nodes and four levels.

Subsection 4.3 discusses the alignment method used to make comparison easier (**R4**). As shown in Figure 2, BCT clearly shows node deletions and insertions as nodes appearing or disappearing within a column. However, if a node is displaced in two trees (i.e., moves from under one parent to another), it is hard to identify such correspondence. With BCT, a displaced node is duplicated and mapped to two columns. Specifically, the node is present in the first column of one tree and the second column of the other tree. An alternative design is to show explicitly visible links between the BCT rows to more clearly indicate node displacements. However, this would come at the cost of increased vertical space requirements and is beyond the focus of this work.

4.2 Interaction Design

To help users understand and navigate the hierarchical data, we designed three interaction techniques for BCTs. In the following, we use

Fig. 4. Diagonal stripe glyphs. (a) Selecting the root nodes of two subtrees of interest; (b) Replacing each contiguous uninteresting region with diagonal stripe glyphs. The density of the diagonal stripe texture in each glyph encodes the number of elided nodes.
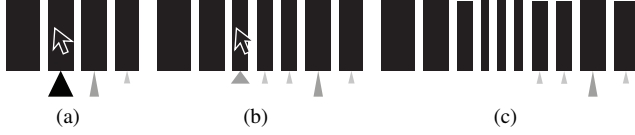

Fig. 5. Collapse/Expand subtrees. The figure above shows the process that users refine their focus of interest. Darker icons correspond to branches with more nodes. Taller icons correspond to deeper branches, and wider icons correspond to a higher average branching factor. As users change the focus of the layout (i.e., click on the node under the cursor in (a)), more details in (b) are revealed. (c) Changing the focus of layout to a deeper branch by clicking on the node under the cursor in (b).

BCT_w as an example to illustrate the interaction techniques and these designs can also be applied to BCT_h.

Structural cue highlighting. Although users can discern the topology in BCTs, this process can be tedious, requiring users to scan nodes sequentially. To help with such interpretation (**R5**), we added interactive structural cues. Figure 3 shows how ancestors, children, and siblings of the node under the cursor are indicated with underlining or tick marks. To indicate descendants, we firstly used a single horizontal stroke (Figure 3(a)). However, we found that analysts still had difficulties in understanding the characteristics of the descendants, such as the number or the balance of subtrees. Therefore we developed the variant in Figure 3(b), where each subtree is indicated with a separate horizontal stroke. The number and lengths of strokes indicate the number and size of subtrees, making it easier to judge whether the tree is balanced.

Compress with diagonal stripe glyphs. For the trees with tens of thousands of nodes, BCTs representation can be very wide and hard to accommodate in a screen. When exploring large hierarchical data, users are usually interested in a few subtrees, and the other parts will distract them. If several subtrees of interest cannot be visualized within the same screen space, users need to scroll horizontally back and forth to navigate between these subtrees. To facilitate such exploration, users can select the subtrees of interest and then aggregate each contiguous uninteresting region into diagonal stripe glyph (Figure 4).

Collapse/Expand Nodes. When the nodes in one branch cannot be visualized within a screen, it will hinder users' navigation. We provide collapse/expand interaction to make the best possible use of the BCT visual representation for interactive visualization. Figure 5 shows the progressive opening of branches as users refine their focus of interest. Compressed branches are previewed with an isosceles triangle. The preview icons are below the root node of the compressed branch.

The visual encodings of the triangular glyph are similar to Space-Tree [51]. The width of the triangle's base encodes the average width (number of items divided by the depth). The height encodes subtrees' depth. The darkness encodes subtrees' total number of nodes. Even for the collapsed subtrees, users can still get overview information from the triangle glyph, and even make rough comparisons of whole subtrees. Clicking on nodes can open collapsed branches.

4.3 Alignment Comparison Method

To compare the topological structures and node attribute values among multiple trees, we developed an alignment comparison method involving two steps. The first step is to juxtapose multiple trees vertically, and the second step is to align the BCTs horizontally. We also introduce visual encodings of alignment results and sorting interactions.

Horizontal alignment. Our method aligns multiple trees and maps the matching nodes to the same horizontal positions to help the user compare their topological structure (**R4**). Our alignment method constructs a *union tree* to merge multiple trees into a single tree (Algorithm 1), then visualize *union tree* with BCT to get *union BCT*, and maps the nodes in each tree to the positions of their matching nodes in the *union BCT*. The *union BCT* is composed of BCTs as a kind of superset of the nodes. Constructing the union BCT is a top-down recursive process. Each iteration merges all matched children (and their subtrees) and then appends unmatched children. The mapping strategy determines the position of each node and allows placing the matching nodes in the same horizontal position after stacking multiple trees vertically.

Algorithm 1 Construction of the union tree

Require: T_1, T_2 are root nodes of two trees to merge.

1: **procedure** BUILDUNIONTREE(T_1, T_2)
2:   **if** T_1.key = T_2.key **then** // if the keys of the root nodes match ...
3:     $C_1[\,] \leftarrow T_1$. children
4:     $C_2[\,] \leftarrow T_2$. children
5:     // $C_{matched}$ is an array of 2-element arrays.
6:     // of children with matching keys:
7:     $C_{matched}[\,] \leftarrow \{[c_1, c_2] | c_1 \in C_1 \wedge c_2 \in C_2 \wedge c_1.\text{key} = c_2.\text{key}\}$
8:     // $C_{unmatched}$ is an array of the remaining children.
9:     $C_{unmatched}[\,] \leftarrow C_1 \cup C_2 - \cup_i C_{matched}[i]$
10:     // C_{union} will be an array of merged subtrees.
11:     $C_{union}[\,] \leftarrow [\,]$ // initially empty
12:     **for** $pair[\,] \in C_{matched}$ **do**
13:       $c_{union} \leftarrow$ BUILDUNIONTREE($pair[0], pair[1]$)
14:       C_{union}. push(c_{union})
15:     **end for**
16:     $T_{union} \leftarrow$ new Node(T_1.key) // give it same key as T_1.
17:     T_{union}.children $\leftarrow C_{union} \cup C_{unmatched}$
18:   **else**
19:     $T_{union} \leftarrow$ new Node(new key) // generate a new key for it.
20:     T_{union}.children $\leftarrow [T_1, T_2]$
21:   **end if**
22:   **return** T_{union}
23: **end procedure**

The alignment result merges the nodes of multiple trees, so the width of the alignment results is larger than single BCT. Our method supports to align a subset of nodes interactively and lay out other nodes successively as the original BCT, which reduces the width of comparison results. Our method provides two alignment strategies, aligning the subtrees of interest and aligning the tree to a certain level.

Figure 6(a) shows three original BCTs. Our method aligns the subtrees of interest by partitioning the BCTs into segments according to whether they need to be aligned. Each partitioned segment contains a single subtree. Then it computes the maximum length of each segment and aligns them at first. Then our method computes the layout of the nodes within each segment separately. As shown in Figure 6(b), for the unaligned segments, our method lays out the nodes based on the BCT layout algorithm. For the aligned segments, our method lays out the nodes using the alignment comparison algorithm based on the *union BCT*.

For aligning the BCTs to a certain level as shown in Figure 6(c), our method firstly constructs the union tree for the nodes above the level, then computes the maximum length for the descendants among the leaf nodes of these trees. Next, it places the nodes of union tree with reserving the blank space, the length of which is the maximum length after each leaf node. Finally, our method computes the positions of the nodes under the aligned level in the corresponding blank space using BCT layout algorithm.

Visual encoding. The alignment results encode the non-existent nodes into outlined rectangles □. Users can select the nodes/subtrees as the reference and identify the extra/missing parts in other trees. The visual encoding of BCT facilitates users to understand the differences.
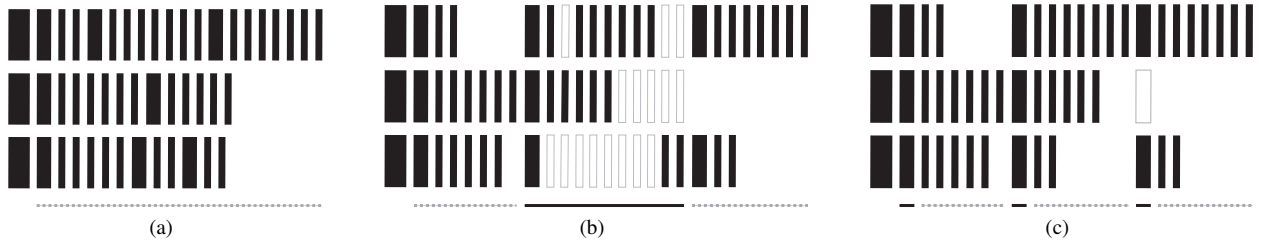
Fig. 6. Node alignment for comparison. (a) Juxtapose the original BCTs vertically; (b) Align the nodes in the subtrees of interest; (c) Align the nodes to the second level. The black line at the bottom of the BCTs indicates the aligned parts, and the gray dotted line indicates the unaligned parts;
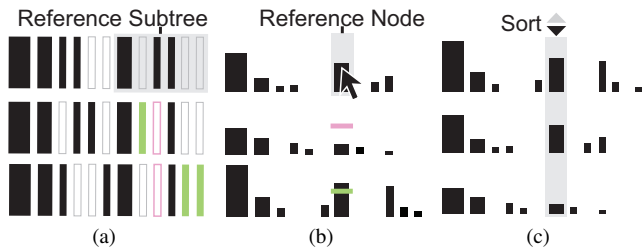

Fig. 7. Visual encoding of the differences between trees. The reference nodes/subtrees are with a light gray background. (a) Topology comparison; The outlined rectangles represent the non-existent nodes. After setting the reference nodes/subtrees, red outlined rectangles encode the missing parts, and green rectangles encodes the extra parts. (b) Node attribute value comparison; After setting the reference node, the red line indicator is added on the matching nodes with a smaller attribute value, and a green line indicator is added on the matching node with a larger attribute value. (c) Sorting BCTs; Three BCTs are sorted descending according to the values of the reference node.

It maps the missing nodes to red outlined rectangles ☐ and the extra nodes to green rectangles ▮ as illustrated in Figure 7(a). Encoding the node attribute value into height might be difficult for comparison between different trees, especially when their differences are small. After selecting the reference as illustrated in Figure 7(b), the indicators will be added on the matching nodes to show the negative and positive changes of node values. The position of the indicators is identical to the node value of reference nodes, and its color indicates the negative or positive differences (**R4**).

Sorting. Users can rank multiple BCTs interactively according to different criteria. Firstly, the user can select the subtrees of interest and rank multiple trees by the node number or topological structure similarities. Secondly, the user can choose to rank multiple trees according to the values of the selected node. To reveal the evolution patterns, the user can rank these trees by their original characteristic, for example, the chronological order.

5 EXPERIMENT

We conducted a controlled experiment to evaluate the performance of the two BCT variants in four different tasks.

5.1 Choice of Techniques

We aim to evaluate the two variants, BCT_w, which is more vertically compact, and BCT_h (Figure 1), which may allow for easier understanding of the topological structure. For experimental comparison, we considered which existing techniques also allow multiple trees to be stacked along the vertical (y) axis, with the nodes of each tree packed along the horizontal (x) axis. More precisely, we considered techniques where the x-coordinate of leaf nodes increases monotonically during a depth-first traversal. This criterion means that we do not consider radial layout techniques [3,21,59,67], as well as inclusion techniques such as treemaps [35,55] and nested circles [8,62].

One well-known technique that *does* arrange leaf nodes along one axis is the icicle plot [40] (Figure 8, left). Classical node-link layout techniques [11,52] also do this and can be obtained by connecting the
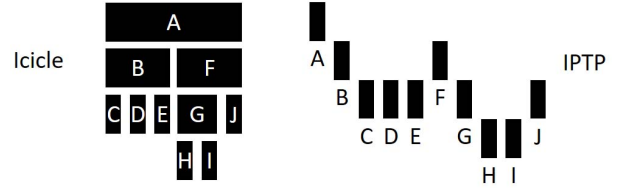

Fig. 8. Competing techniques for experimental evaluation.

centers of the rectangles in an icicle plot with line segments along with parent-child relationships. However, compared to classical node-link layouts, the rectangles in icicle plots provide more room for labels and are easier to select with a mouse. Icicle plots have been used previously for comparing multiple trees [61], although without alignment.

Another candidate technique is the Indented Pixel Tree Plot (IPTP) (Figure 8, right). Previous work [13] on this does not use it to compare multiple trees, but IPTP *could* be used in this way, and has the advantage that it assigns a unique x position to each node (not just to leaf nodes), allowing matching nodes of stacked trees to be aligned and compared. Note that the originally published form of IPTP renders leaf nodes in a smaller size, however this would hinder the display of attributes on the leaf nodes, hence our implementation draws all nodes the same size.

For each technique, we can ask: does each child node overlap its parent node (1) along the x axis and (2) along the y axis? With icicle plots, the answers to these questions are (yes, no), that is to say, there *is* overlap along the x axis but *not* along the y axis. With BCT, the answers are (no, yes). With IPTP, the answers are (no, no). Each of the two questions offers a tradeoff, with overlap saving space but possibly making it more difficult to distinguish nodes.

Thus, four techniques were implemented and investigated experimentally: BCT_w, BCT_h, ICICLE, and IPTP. As we explain later, IPTP was eliminated after our pilot study and not included in the full experiment.

5.2 Analysis of Techniques and Hypotheses

Table 1 compares the four techniques according to several criteria. The 3rd row concerns vertical space requirements. If we assume that all techniques display leaf nodes of the same size, then clearly BCT_h and especially BCT_w save space vertically over the other techniques, allowing different trees to be stacked closer together. The 4th row concerns the size of nodes: ICICLE results in larger non-leaf nodes, making them easier to select with a mouse. The last two rows in the table are grey because our answers are tentative and hypothetical. The second-last row indicates which techniques are best at communicating the structure of the tree: we suspect that BCT_h and especially BCT_w will suffer here.

The last row concerns the comparison of multiple trees. We suspect ICICLE does not support easy comparison of the matching nodes in different trees, as the intervening space between matching nodes is filled with other nodes, which hinders finding the matching nodes. IPTP, however, assigns a unique x position to each node, facilitating comparison of matching nodes in different trees, as there are no intervening nodes. Both BCT techniques also assign unique x positions to each node, and also allow different trees to be closer together than the

Table 1. Comparison of techniques

	BCT_w	BCT_h	ICICLE	IPTP
Can encode attribute with color	yes	yes	yes	yes
Can encode attribute with height	yes	no	yes	yes
Saves space vertically	yes	partially	no	no
Non-leaf nodes are bigger, hence easier to select	no	no	yes	no
Easy to understand structure of single tree	no	maybe	yes	yes
Easy to compare nodes in different trees	yes	yes	no	maybe

other two techniques, which could make a comparison even easier.

The last two rows in the table are the ones we seek to investigate experimentally. We aim to confirm that BCT is indeed better for comparing matching nodes, and also test if other techniques are better at helping users understand the structure of trees. We formulate the following hypotheses:

H1 - BCT will require more time and/or result in a higher error rate than ICICLE or IPTP for tasks requiring the understanding of the structure (topology).

H2 - BCT will require less time than ICICLE or IPTP for tasks requiring the comparison of multiple trees.

H3 - BCT will result in an error rate that is competitive with (i.e., no worse than) ICICLE or IPTP in tasks requiring the comparison of multiple trees.

5.3 Experiment tasks

To test the preceding hypotheses, we chose two tasks (Task1, Task2) that require the user to understand the structure of a single tree and two tasks (Task3, Task4) that require comparing multiple trees. Examples are illustrated in Figure 9.

Task1 (Find a descendant node): A single tree is displayed, within which one node N is highlighted. The user must find the second child of N's second child, then click this node to finish the task.

Task2 (Find the nearest common ancestor): A single tree is displayed, within which two nodes N_1, N_2 are highlighted. The user must find the nearest (i.e., deepest) common ancestor of N_1, N_2 and click it. If N_1 is itself an ancestor of N_2, the user must click on the parent of N_1.

Task3 (Tree comparison without scrolling): The alignment results of three trees are displayed within the screen space, stacked vertically to place the matching nodes at the same horizontal positions, with one tree T_1 highlighted to indicate that is the "reference tree". The user must compare the trees to find, among the other two trees, which one (call it T_2) contains the largest number of identical nodes with the reference tree T_1. The user must click on the label of T_2 to finish the task. Note that this task does not involve clicking on a node, but rather on a label for an entire tree, and that these labels were always the same size regardless of the tree layout technique used.

Task4 (Tree comparison with scrolling): This is the same as Task3, except that now there are six trees to compare, and only three are visible at a time within the screen space, requiring the user to scroll vertically to compare the trees. Scrolling was done using the mouse wheel (i.e., the user did not need to click on a scrollbar to begin scrolling firstly). To complete the task, the user had to click on the label of one of the five non-reference trees to indicate which had the largest number of identical nodes with the reference tree.

Tasks 1 and 2 are designed to test hypothesis H1, while Task3 and Task4 measure ability to compare sets of nodes across multiple trees to test H2 and H3. Task3 and Task4 are designed to measure the benefit of saving vertical space, especially Task4, where ICICLE and IPTP require users to scroll more.

Task1 is similar to tasks in previous studies [38, 58] where users had to find a node with particular properties. Task2 has been used in other previous studies [4, 12]. Tasks 3 and 4 are similar to Wang et al.'s [66] which asked users to find subtrees that are "very similar", "slightly similar", "significantly different", etc.

For all tasks, the maximum vertical size of nodes was always set to be the same, and the horizontal spacing between the adjacent nodes was the same as well. Furthermore, with the BCT_h and ICICLE techniques, the width of leaf nodes was always the same. With BCT_w, however,
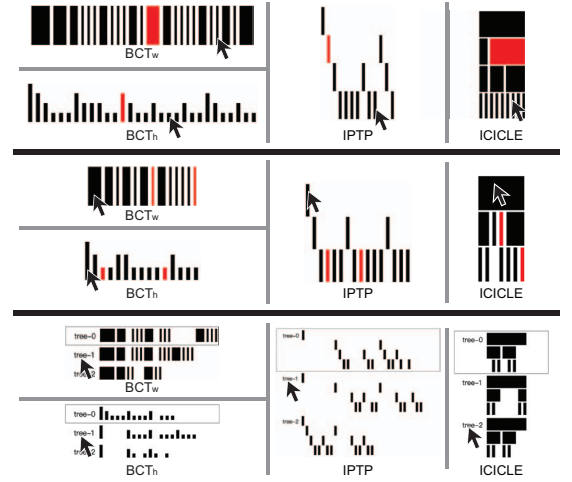

Fig. 9. Example tasks in the controlled experiment. Nodes highlighted at the start of the trial are red, and cursors above show where the user had to click. (Top) Task1: Click the second child of the second child of the highlighted node. (Middle) Task2: Click the nearest common ancestor of the highlighted nodes. (Bottom) Task3: Click the tree most similar to the reference tree (tree-0) indicated by the box.

leaf nodes were thinner, because BCT_w requires having thin leaf nodes to allow for a range of widths for non-leaf nodes.

None of our tasks required the user to compare attribute values between different nodes. This is because previous work [31] has already studied comparisons of pairwise attributes encoded with different channels. We were instead interested in understanding the tradeoffs in using BCT to understand tree structure and compare multiple trees.

5.4 Pilot study

A pilot study compared all four techniques (BCT_w, BCT_h, ICICLE, and IPTP) with four users and all four tasks. The users were from the computer science department. The goal of the pilot was to validate if the experimental design is appropriate. During the pilot, each user took more than 1.5 hours to complete all the tasks. The main result of the pilot was that IPTP was always slower than ICICLE, in all tasks. Before the pilot, we suspected that IPTP could benefit over ICICLE by not having any nodes in the intervening space between matching nodes. However, during interviews, users reported that the nodes occupying intervening space (in ICICLE) do not hinder comparison; on the contrary, they provide references to help find matching nodes in other trees. Based on the measured times and interview results, we removed IPTP from the full experiment, thus allowing users in the full experiment to complete all tasks within one hour.

5.5 Full experiment

To understand the advantages and disadvantages of BCT, we conducted a full experiment to compare BCT_w, BCT_h, and ICICLE.

Participants and Apparatus. Twenty-one users (aged 25-35) from both local universities and industry participated. The users were from the fields of computer science, economics, business, and finance. All of them were right-handed. All the subjects indicated that they were familiar with hierarchical data and had experience with visual analysis tasks. Tasks were performed in a quiet computer lab on a Dell Precision T5500 desktop PC, with an Intel Xeon Quad-Core processor, 8GB RAM, and an NVidia Quadro 2000 graphics card driving a 23IN LCD 1920×1080 pixel monitor. The experiment system was developed in JavaScript using Node.js.

Dataset. We decided to use algorithmically-generated data, rather than real-world data, so as to more easily control the characteristics of the data, making it easier to replicate a similar study in the future (if so desired) and avoiding the bias that would come from using any particular real-world dataset. (As a complementary approach, the qualitative evaluation in Section 6 uses real-world data.)
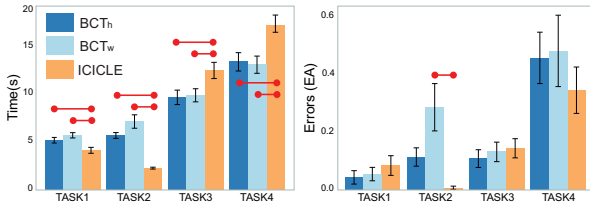
Fig. 10. Experiment results are broken down by the techniques and tasks. Vertical error bars indicate standard errors. Horizontal red line segments indicate pairs that are significantly different ($p < 0.05$).

Trees for the experiment were generated such that each tree or set of trees fits within the screen without scrolling, except in Task4. Each tree had a depth between 2 to 5, and each node had between 0 to 5 children. The differences between trees of the same set involved nodes (or subtrees) either appearing (insertions) or disappearing (deletions), but not nodes moving from one location in a tree to another (displacements).

The BCT layout can show node displacements by duplicating each displaced node in the alignment results (discussed in Section 4.1). However, the data generated for the experiment did not involve such node displacements, for two reasons.

Firstly, none of the techniques in the experiment use explicit links to show matching nodes between trees, in order to save vertical space. Thus, to show node displacements, it would be reasonable for all the techniques to use node duplication. Furthermore, all four techniques assign x coordinates to leaf nodes in the same depth-first order. These similarities suggest that all the techniques would suffer similarly when used with data involving node displacements, and generating such data for the experiment would likely only prolong the trials and/or reduce statistical power.

Second, the real-world datasets used for qualitative evaluation of the prototype (Section 6) are based on a library book classification where nodes never move from one place to another. This is at least one real-world domain where node displacements simply do not occur.

Procedure. A within-subjects design was used, with each user completing all four tasks with each of the three techniques. Tasks were completed in a fixed order. Within each task, users underwent each of the three techniques in counterbalanced order, beginning with warm-up trials. In total, there were 21 users × 4 tasks × 3 techniques × 8 repetitions = 2016 trials.

Users were instructed to complete each trial as accurately and as quickly as possible and were told that accuracy and completion time are equally important. They were seated ≈ 80 cm from the monitor.

Experiment results. Times and error rates (Figure 10) for each task were analyzed using repeated measures ANOVA ($\alpha = 0.05$) with Bonferroni adjusted post-hoc comparisons.

Task1 (Find a descendant node): technique had a significant effect on time ($F_{2,501} = 7.9$, $p < 0.0005$). Pairwise t-tests showed that ICICLE was significantly faster ($p < 0.05$) than each of the BCT techniques, whereas the two BCT techniques did not significantly differ ($p > 0.05$). A subsequent ANOVA found no significant difference between the error rates of the techniques ($p > 0.05$).

Task2 (Find the nearest common ancestor): technique had a significant effect on time ($F_{2,501} = 31.32$, $p < 10^{-12}$). Pairwise t-tests found that ICICLE is significantly faster ($p < 10^{-6}$) than each of the BCT techniques, whereas the two BCT techniques are only weakly different ($p = 0.061$). A subsequent ANOVA showed that the technique also had a significant effect on error rate ($F_{2,501} = 7.62$, $p < 0.001$). Pairwise t-tests found that ICICLE only had a significantly lower error rate than BCT_w ($p < 0.0005$), whereas the two BCT techniques are only weakly different ($p = 0.052$).

Task3 (Tree comparison without scrolling): ANOVA found that technique had a significant effect on time ($F_{2,501} = 4.32$, $p < 0.02$). Pairwise t-tests showed that ICICLE is significantly slower ($p < 0.05$) than each of the BCT techniques, whereas the two BCT techniques do not significantly differ ($p > 0.05$). There were no significant differences in error rate ($p > 0.05$).

Task4 (Tree comparison with scrolling): ANOVA again found that technique has a significant effect on time ($F_{2,501} = 5.42$, $p < 0.005$). Pairwise t-tests showed that ICICLE is significantly slower ($p < 0.02$) than each of the BCT techniques, whereas the two BCT techniques do not significantly differ ($p > 0.05$). There were no significant differences in error rate ($p > 0.05$).

Discussion of results. Given the results of the pilot study, we remove consideration of IPTP and slightly rephrase the hypotheses as follows. H1: the BCT techniques require significantly more time than ICICLE for Task1 and Task2. H2: the BCT techniques require significantly less time than ICICLE for Task3 and Task4. H3: the BCT techniques yield error rates not significantly worse than ICICLE for Task3 and Task4. Formulated like this, all three hypotheses are confirmed. As expected, the BCT techniques make topological interpretation tasks (Task1 and Task2) more difficult, but makes tree comparison tasks (Task3 and Task4) easier, probably because the design of BCT allows the distance between trees to be reduced. As explained in Section 3, showing parent-child relationships (**R5**) is the least important design requirement for BCT, in the interest of showing as many trees and nodes as possible. In tasks requiring more topological interpretation, it could be important for the user to be able to switch to alternative visualizations that make the topological structure more clear.

We also note that all four tasks required users to first determine the answer to the question being asked (e.g., find the nearest common ancestor), and then to move the mouse cursor to an on-screen target and click. The time required for moving the mouse cursor and clicking is governed by Fitts' law [45]. This robust predictive model has been confirmed in hundreds of studies [1] and predicts that smaller targets require more time to point and click on. In Task3 and Task4, the targets clicked by the user were always the same size, thus giving no advantage to any technique. However, in Task1 and Task2, the user had to click on a node of the tree, and the non-leaf nodes of ICICLE are bigger than the nodes of BCT layouts. Therefore we should expect ICICLE to have a greater time advantage in these tasks, especially in Task2 where users *always* had to click on a non-leaf node, often a very shallow node (and therefore a node that is much larger in ICICLE). Indeed, the experimental results show that ICICLE had the biggest time advantage in Task2. Because Task1 and Task2 measured the combined time to *understand* which node to click on, and also the time to move the mouse cursor to that node and click, the measured times do not reflect the true difference in time to understand topology. In other words, BCT is not as bad at topological understanding as the results suggest.

Finally, comparing the two BCT variants, it is interesting that BCT_h required less time and resulted in a lower error rate than BCT_w for Task1 and Task2, though not significantly. This is not surprising, since the design of BCT_h is intended to make the depth of each node clearer. However, this design comes at the cost of making BCT_h less vertically compressible than BCT_w, and also meaning that node attribute values can only be shown using color, which is less effective than using height for quantitative values [31].

6 QUALITATIVE EVALUATION

We demonstrate the utility of the BCT technique via two use cases that were identified by working with two domain experts. We also report feedback from these experts during follow-up interviews.

6.1 Dataset

Our evaluation uses the records of borrowed books at a university library. Libraries often organize books according to the *Dewey decimal classification*, a proprietary, hierarchical system. Deeper levels of the tree classify books with ever-increasing refinement. The root node contains all the books, and its children correspond to *literature*, *technology*, *social science*, etc. The *social science* node has children for *political science*, *economics*, *law*, etc. Each leaf node represents the most granular book category.

The dataset contains the records of borrowed books for two years (01/01/2016 to 31/12/2017). For each day, we constructed a tree based on the Dewey decimal classification of all the borrowed books, where each node's attribute value is equal to the number of borrowed books in that node's category. The dataset contains 730 trees, one for each day.
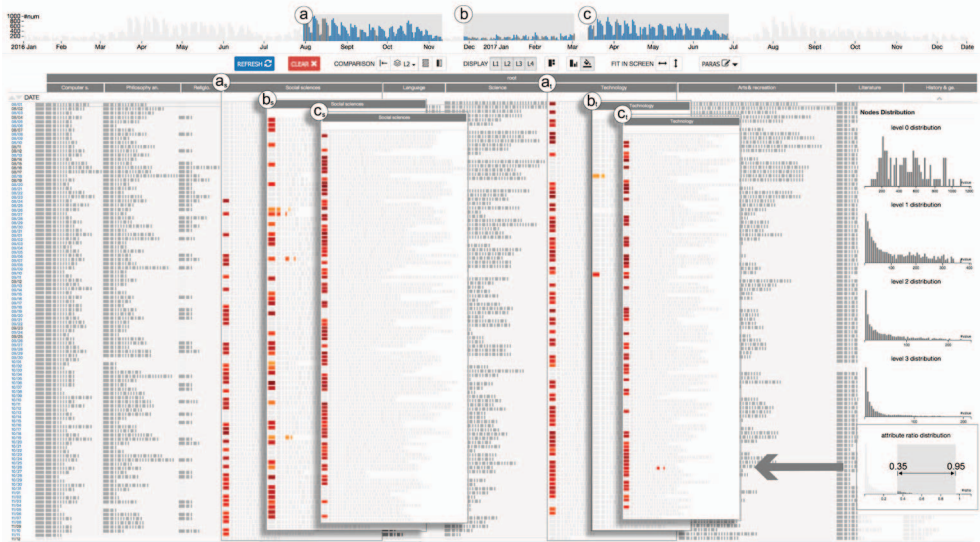
Fig. 11. The histogram along the top shows the number of borrowed books over the entire two years. The user firstly selects interval (a), yielding the view in the main view (shown as a background layer in the figure) of just over 100 days. In other words, just over 100 trees (with hundreds of nodes per tree) are visible simultaneously. The different columns correspond to categories of books, including columns (a_s, a_t), where subscripts indicate the category: s for *Social Science*, t for *Technology*. Later, the user selects intervals (b) and (c), resulting in the data shown in columns (b_s, b_t, c_s, c_t), shown in the figure as cropped layers superimposed on the original main view for comparison. Throughout all these views, the user has set the ratio filter in (d) to the range 0.35 to 0.95 to show only the categories that account for the most commonly borrowed books, and not including the root whose ratio is 1.0. In interval (b), the activity in *Technology* (b_t) is clearly different.

6.2 Prototype

We have implemented a prototype system based on BCT techniques. In this system, an overview histogram on the top (Figures 12 and 11) shows the sum of borrowed books (the attribute values on the root node) for each day. Users can select intervals within the histogram. Below the bar chart, interactive views support comparison of multiple BCTs. We have also implemented the interaction techniques proposed in Section 4, including *structural cues*, *diagonal stripe glyphs*, and *node alignment*. The system also provides auxiliary functions for data exploration, including *filtering* and *sorting*.

6.3 Expert Users

We invited two university librarians, each having at least 15 years of experience, to evaluate our prototype system for comparison of multiple trees based on BCT. The two experts are interested in comparing the borrowing records of different time to optimize the allocation of human resources and book management. Before the exploration of the dataset, the experts are given a brief introduction of the usage of the prototype system. The followings are two use cases that emerged from their exploration. The first case shows that BCTs can support to compare the topological structure of multiple trees, and the second case shows the BCTs can support to compare the node attribute values.

6.4 Case 1: Topological structure comparison

To explore the differences between the book borrowing patterns of two semesters in one year, the experts brushed the overview histogram and selected two similar periods in two semesters: interval (a) (from Mar. 20 to Apr. 22 in 2017) and interval (b) (from Jul. 21 to Aug. 21 in 2017), as shown in Figure 12. From the overview histogram, the experts learned that the amounts of borrowed books are similar. The selected trees are visualized as BCT rows and stacked vertically for comparison. The experts aligned the subtrees at the second level to place the matching nodes at the same horizontal positions. They identified that the number of borrowed books in the *Language* category of the first period is much larger than that of the second period. The experts then focused on the *Language* category, compressed the other parts with the diagonal stripe glyphs, and aligned the subtree of the *Language* category at the third level. The exploration results at the bottom of Figure 12 reveal the different topology variation patterns.
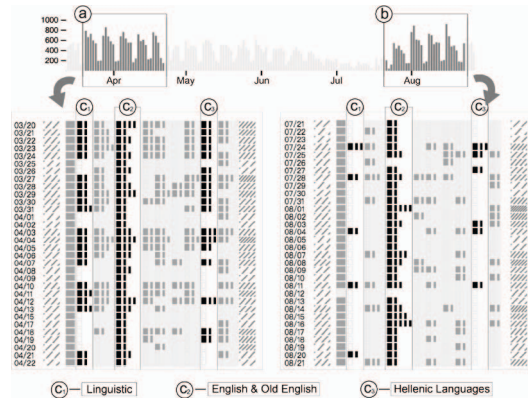

Fig. 12. Changes in books borrowed over time. Within the overview histogram along the top, the user has selected intervals (a) and (b), which are similar periods in different semesters. The two intervals are visualized below (in the left and right panes, respectively) of the main view. Both panes focus on the *Language* subtree, replacing other subtrees with diagonal stripe textures. Matching nodes are aligned. Three categories under *Language* are labelled c_1, c_2, and c_3. In both intervals, c_2 (*English and Old English*) shows similar continuous activity. In interval (a) and interval (b), c_1 (*Linguistics*) exhibits different periodic patterns. Specifically, students borrowing book nearly every weekday in interval (a) and on every Monday in interval (b).

Firstly, at the overview level, the experts found that students borrow books in the *Language* category more frequently in interval (a) compared with interval (b). The experts also noticed that the topological structure changes of the category *English and Old English* in both two intervals are quite small. Specifically, the borrowing records of this category exist almost every day. However, the borrowing patterns of the categories *Linguistic* and *Hellenic Languages* in interval (a) are quite different from that in interval (b). Furthermore, the experts continued to compare the topological structure constructed by the borrowing records of different days. For the *Linguistic* category under interval (a), the experts found that the students usually do not borrow the books of this category at weekends (identifying the missing nodes). For the

Linguistic category under interval (b), they found that students usually borrow books (identifying the extra nodes) on Monday.

After checking the university handbook, the experts found that many foundation courses for international students are in the first semester. Therefore the books related to the *Language* categories is more popular in the first semester. From the detailed borrowing records, experts also found that the international students borrowed most books under the *Language* categories. With this discovery, the librarians can change the location of the books under the *Language* categories to more handy places and add more administrators and volunteers to help the international students to find these books in the first semester.

6.5 Case 2: Node attribute comparison

The second case explores the differences between students' borrowing behaviors between term-time and vacations. The experts want to find the node attribute variation patterns of different categories. As shown in Figure 11, the experts first brushing two semesters in the overview histogram, interval (a) (from Aug. 1 to Nov. 12 in 2016) and interval (c) (from Mar. 13 to Jul. 1 in 2017). The experts then start their explorations from the comparison results overview. Our prototype provides filtering interactions according to the ratio between the amount of borrowing records under this category and the total number of the borrowing records. As the experts want to explore the most frequently borrowed categories, they set the ratio in the range of 0.35 to 0.95. The filtered nodes have a relatively large amount of borrowing records. The visualization results showed that in interval (c), *Social Science* and *Technology* have a large amount of borrowed books nearly every day (c_s and c_t). In interval (a), the experts found that the number of borrowing records has a rising trend in the *Social Science* category (a_s). Specifically, fewer nodes are filtered out at the beginning, and then it appears more nodes along the time. However, borrowing records in the *Technology* category (a_t) stays stable. Next, the experts selected the vacation period interval (b) (from Nov. 28 in 2016 to Mar. 2 in 2017), and adjusted ratio controller to filter the ratio of numbers of book borrowing. From the overview, the experts found a different attribute variation patterns between the *Social Science* category and the *Technology* category (b_s against b_t). Specifically, a large number of borrowing records lie in *Social Science* during interval (b). However, the number of records in the *Technology* category is small. Furthermore, the experts also found two abnormal increases in the middle.

For the explanation of this variation pattern, the handbook indicates that the university provides many elective subjects in interval (b) from different departments. Students need to select the courses of interest and count into their credits. Then, the experts looked into the detailed information of the courses related to *Social Science* and *Technology*. There were 17 courses in *Social Science* which with 11 courses taught in the classroom. However, students related to *Technology* only had five courses in the classroom, and other 16 courses were involved in industrial practice and internship. Only the courses that need to take lessons in the university requires students to borrow books from the library. In this way, the students are more likely to choose these courses related to *Social Science* than *Technology*. As described above, the course arrangements in interval (a) and (b) are very different, but the courses in interval (a) and (c) do not have a significant difference between *Social Science* and *Technology*.

6.6 Expert Feedback

We conducted one-on-one 45-minute interviews with the two domain experts to collect feedback on our techniques. We asked the experts to explore on their own using our prototype. During this process, we answered their questions and recorded comments and preferences. Both experts expressed enthusiasm toward using BCT. "BCT technique is a little hard to understand at first, but it is easy to learn and use" (E1) and "It provides interactive tools to help me explore data with full confidence" (E2). E1 and E2 also confirmed that our search strategies and interactive suggestion could help them to recognize variation patterns. E2 is a technical staff in the library, and he deemed that BCT techniques were good methods to save space which allows stacking multiple trees vertically. "The prototype system with interactive auxiliary tools can facilitate decision maker for forecasting, preparing and tracking future

activities" (E1). They agreed that the BCT techniques can be easily applied to other analysis tasks involving multiple trees.

7 CONCLUSIONS AND FUTURE DIRECTIONS

We have presented BarcodeTree, the first visualization technique for multiple *shallow* and *stable* trees that enables comparison of topological structure and node attribute values of one hundred trees on a single screen (Figure 11). Our experimental evaluation found that, BCT requires significantly more time than ICICLE for understanding the topology of a single tree (Task1 and Task2), but requires significantly less time than ICICLE for the comparison of multiple trees (Task3 and Task4). We also demonstrated the utility of BCT through case studies with real-world data comprising hundreds of trees.

Future work could design ways to easily switch between a BCT overview and other visualizations of trees that make topology easier to understand, to combine the best of both approaches. Our experimental results also suggest that the BCT_h variant makes it easier to understand topology than BCT_w but at the cost of being less vertically compressible than BCT_w. Future work could design a method to smoothly animate between these two variants, allowing the user to transition to one or the other according to current needs.

Some limitations of BCT are that they become less readable if some nodes have a large fan-out [58], and BCT is not designed for comparing a small number of trees that each have large numbers of nodes. This might also be addressed in future work.

BCT is designed to save space vertically and therefore does not show explicitly visible links between matching nodes across trees. Because of this, if there are node displacements across trees, BCT can only show this by duplicating the node in the alignment results (Section 4.1). Future work could evaluate the tradeoffs of such node duplication versus showing explicit links, perhaps taking inspiration from an earlier study involving network data [32]. Future work could also evaluate a hybrid visualization that only sometimes, or only optionally, displays explicit links between consecutive trees, to reduce the vertical space required. The use of explicit links would also be helpful in cases where the trees to compare are very different, and trying to align matching nodes would result in a very sparse visualization.

Tree comparison requires associating each node in one tree with the matching node in another tree. However, our work does not focus on matching algorithms, and we assumed that each node in the hierarchical data has one unique key, so the nodes with the same key in multiple trees can be perfectly matched in linear time [19]. TreeJuxtaposer [48] instead computes the best corresponding nodes through pairwise similarity scores, which require quadratic time to compute. There is still research left to be done on tree matching algorithms.

Regarding the visual design of our techniques, the glyphs with diagonal stripe texture cannot reflect the node number precisely, but they are designed to reduce the width of context and help users focus on the subtrees. They can also provide users with hints to support further exploration. When visualizing a very deep tree or compressing the BCT significantly, the width/height difference between the nodes at the adjacent levels will be tiny, and it might be difficult for users to discern the topological structure. In these cases, users are allowed to select a given level of granularity and update the width/height interactively. Our method supports encoding the numerical node value and compares them among multiple hierarchical data. Future work could investigate the comparison of multivariate attributes over multiple trees.

Our method provides users with an overview of visualizing multiple trees within the screen space. From the overview, users can also learn the detailed information of each node. However, users might be overwhelmed with this information and find it difficult to explore. In the future, we plan to explore automatically pattern detection algorithms based on BCT alignment results to facilitate users' exploration.

REFERENCES

[1] http://www.yorku.ca/mack/rn-fitts_bib.htm.

[2] N. Amenta and J. Klingner. Case study: visualizing sets of evolutionary trees. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 71–74, 2002.

[3] K. Andrews and H. Heidegger. Information slices: Visualising and exploring large hierarchies using cascading, semi-circular discs. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 9–12, 1998.

[4] T. Barlow and P. Neville. A comparison of 2-D visualizations of hierarchies. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 131–138, 2001.

[5] F. Beck and S. Diehl. Visual comparison of software architectures. *Information Visualization*, 12(2):178–199, 2013.

[6] F. Beck, J. Melcher, and D. Weiskopf. Identifying modularization patterns by visual comparison of multiple hierarchies. In *IEEE Int. Conf. Program Comprehension (ICPC)*, pp. 1–10, 2016.

[7] F. Beck, F. Wiszniewsky, M. Burch, S. Diehl, and D. Weiskopf. Asymmetric visual hierarchy comparison with nested icicle plots. In *Joint Proceedings of the Fourth International Workshop on Euler Diagrams and the First International Workshop on Graph Visualization in Practice co-located with Diagrams*, pp. 53–62, 2014.

[8] R. Boardman. Bubble trees: The visualization of hierarchical information structures. In *Extended Abstacts of ACM Conf. Human Factors in Computing Systems*, pp. 315–316, 2000.

[9] S. Bremm, T. von Landesberger, M. Hess, T. Schreck, P. Weil, and K. Hamacherk. Interactive visual comparison of multiple trees. In *Proc. IEEE Symp. Visual Analytics Science And Technology (VAST)*, pp. 31–40, 2011.

[10] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *Proc. Eurographics / IEEE VGTC Conference on Visualization (EuroVis)*, pp. 33–42, 1999.

[11] C. Buchheim, M. Jünger, and S. Leipert. Improving Walker's algorithm to run in linear time. In *Proc. Symp. Graph Drawing (GD)*, pp. 344–353, 2002.

[12] M. Burch, N. Konevtsova, J. Heinrich, M. Hoeferlin, and D. Weiskopf. Evaluation of traditional, orthogonal, and radial tree diagrams by an eye tracking study. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2440–2448, 2011.

[13] M. Burch, M. Raschke, and D. Weiskopf. Indented pixel tree plots. In *Proc. Int. Symp. Advances in Visual Computing (ISVC)*, pp. 338–349, 2010.

[14] S. K. Card, B. Suh, B. A. Pendleton, J. Heer, and J. W. Bodnar. TimeTree: Exploring time changing hierarchies. In *Proc. IEEE Symp. Visual Analytics Science And Technology (VAST)*, pp. 3–10, 2006.

[15] F. Chevalier, D. Auber, and A. Telea. Structural analysis and visualization of C++ code evolution using syntax trees. In *Int. Workshop on Principles of Software Evolution (IWPSE)*, pp. 90–97, 2007.

[16] E. H. Chi, J. Pitkow, J. Mackinlay, P. Pirolli, R. Gossweiler, and S. K. Card. Visualizing the evolution of web ecologies. In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 400–407, 1998.

[17] N. Churcher, L. Keown, and W. Irwin. Virtual worlds for software visualisation. In *Software Visualisation Workshop (SoftVis)*, pp. 9–16, 1999.

[18] W. Cui, S. Liu, Z. Wu, and H. Wei. How hierarchical topics evolve in large text corpora. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2281–2290, 2014.

[19] W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.

[20] K. Dinkla, M. A. Westenberg, H. Timmerman, S. A. van Hijum, and J. J. van Wijk. Comparison of multiple weighted hierarchies: visual analytics for microbe community profiling. *Computer Graphics Forum*, 30(3):1141–1150, 2011.

[21] P. D. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[22] S. Fu, H. Dong, W. Cui, J. Zhao, and H. Qu. How do ancestral traits shape family trees over generations? *IEEE Transactions on Visualization and Computer Graphics*, 24(1):205–214, 2018.

[23] M. Gleicher. Considerations for visualizing comparison. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):413–423, 2018.

[24] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, 2011.

[25] M. Glueck, A. Gvozdik, F. Chevalier, A. Khan, M. Brudno, and D. Wigdor. PhenoStacks: Cross-sectional cohort phenotype comparison visualizations.

[26] *IEEE Transactions on Visualization and Computer Graphics*, 23(1):191–200, 2017.

[26] M. Glueck, P. Hamilton, F. Chevalier, S. Breslav, A. Khan, D. Wigdor, and M. Brudno. PhenoBlocks: Phenotype comparison visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):101–110, 2016.

[27] M. Graham and J. Kennedy. Exploring multiple trees through DAG representations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1294–1301, 2007.

[28] M. Graham and J. Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010.

[29] J. Guerra-Gómez, M. L. Pack, C. Plaisant, and B. Shneiderman. Visualizing change over time using dynamic hierarchies: TreeVersity2 and the StemView. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2566–2575, 2013.

[30] J. A. Guerra-Gómez, A. Buck-Coleman, C. Plaisant, and B. Shneiderman. TreeVersity: Interactive visualizations for comparing two trees with structure and node value changes. In *Proc. Conf. Design Research Society (DRS)*, vol. 2, pp. 640–653, 2012.

[31] J. Heer and M. Bostock. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 203–212, 2010.

[32] N. Henry, A. Bezerianos, and J.-D. Fekete. Improving the readability of clustered social networks using node duplication. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1317–1324, 2008.

[33] D. M. Hillis, T. A. Heath, and K. S. John. Analysis and visualization of tree space. *Systematic Biology*, 54(3):471–482, 2005.

[34] D. Holten and J. J. van Wijk. Visual comparison of hierarchically organized data. In *Proc. Eurographics / IEEE VGTC Conference on Visualization (EuroVis)*, pp. 759–766, 2008.

[35] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. IEEE Visualization (VIS)*, pp. 284–291, 1991.

[36] B. Kerr. Thread arcs: an email thread visualization. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 211–218, 2003.

[37] D. E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*, pp. 309–310. Addison-Wesley, 1968.

[38] A. Kobsa. User experiments with tree visualization systems. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 9–16. IEEE, 2004.

[39] W. Köpp and T. Weinkauf. Temporal treemaps: Static visualization of evolving trees. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):534–543, 2019.

[40] J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.

[41] D. O. Kutz. Examining the evolution and distribution of patent classifications. In *Proc. Int. Conf. Information Visualisation (IV)*, pp. 983–988, 2004.

[42] B. Lee, G. G. Robertson, M. Czerwinski, and C. S. Parr. CandidTree: visualizing structural uncertainty in similar hierarchies. *Information Visualization*, 6(3):233–246, 2007.

[43] J. Lukasczyk, G. Weber, R. Maciejewski, C. Garth, and H. Leitte. Nested tracking graphs. *Computer Graphics Forum*, 36:12–22, 2017.

[44] R. Lutz, D. Rausch, F. Beck, and S. Diehl. Get your directories right: From hierarchy visualization to hierarchy manipulation. In *IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 25–32, 2014.

[45] I. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7(1):91–139, 1992.

[46] M. J. McGuffin and J.-M. Robert. Quantifying the space-efficiency of 2D graphical representations of trees. *Information Visualization*, 9(2):115–140, 2010.

[47] T. Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.

[48] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. *ACM Transactions on Graphics*, 22(3):453–462, 2003.

[49] P. Neumann, S. Schlechtweg, and S. Carpendale. ArcTrees: Visualizing relations in hierarchical data. In *Proc. Eurographics / IEEE VGTC Conference on Visualization (EuroVis)*, pp. 53–60, 2005.

[50] B. Ondov, N. Jardine, N. Elmqvist, and S. Franconeri. Face to face: Evaluating visual comparison. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):861–871, 2019.

[51] C. Plaisant, J. Grosjean, and B. B. Bederson. SpaceTree: Supporting ex-

ploration in large node link tree, design evolution and empirical evaluation. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 57–64, 2002.

[52] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, 1981.

[53] H. Schulz, S. Hadlak, and H. Schumann. The design space of implicit hierarchy visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):393–411, 2011.

[54] H. J. Schulz. Treevis.net: A tree visualization reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, 2011.

[55] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992.

[56] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Symp. Visual Languages*, pp. 336–343, 1996.

[57] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 73–78, 2001. doi: 10.1109/INFVIS.2001.963283

[58] H. Song, B. Kim, B. Lee, and J. Seo. A comparative evaluation on tree visualization methods for hierarchical structures with large fan-outs. In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 223–232, 2010.

[59] J. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 57–65, 2000.

[60] Y. Tanaka, Y. Okada, and K. Niijima. Treecube: visualization tool for browsing 3D multimedia data. In *Proc. Int. Conf. Information Visualisation (IV)*, pp. 427–432, 2003.

[61] A. Telea and D. Auber. Code flows: Visualizing structural evolution of source code. *Computer Graphics Forum*, 27(3):831–838, 2008.

[62] S. T. Teoh and K.-L. Ma. RINGS: A technique for visualizing large hierarchies. In *Proc. Symp. Graph Drawing (GD)*, pp. 268–275, 2002.

[63] Y. Tu and H. W. Shen. Visualizing changes of hierarchical data using treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1286–1293, 2007.

[64] E. R. Tufte. *Beautiful evidence*. Graphics Press Cheshire, 2006.

[65] C. Vehlow, F. Beck, and D. Weiskopf. Visualizing dynamic hierarchies in graph sequences. *IEEE Transactions on Visualization and Computer Graphics*, 22(10):2343–2357, 2016.

[66] Y. Wang, S. T. Teoh, and K.-L. Ma. Evaluating the effectiveness of tree visualization systems for knowledge discovery. In *Proc. Eurographics / IEEE VGTC Conference on Visualization (EuroVis)*, pp. 67–74, 2006.

[67] J. Yang, M. O. Ward, and E. A. Rundensteiner. InterRing: An interactive tool for visually navigating and manipulating hierarchical structures. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 77–84, 2002.

[68] N. Ytow. Taxonaut: an application software for comparative display of multiple taxonomies with a use case of gbif species api. *Biodiversity Data Journal*, (4):e9787, 2016.

[69] J. Zhao, F. Chevalier, C. Collins, and R. Balakrishnan. Facilitating discourse analysis with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2639–2648, 2012.