

CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies

Taowei David Wang¹ and Bijan Parsia²

¹ Department of Computer Science,
University of Maryland, College Park, MD 20742, USA
tw7@cs.umd.edu

² The University of Manchester, UK
bparsia@cs.man.ac.uk

Abstract. OWL ontologies present many interesting visualization challenges. Here we present CropCircles, a technique designed to view the class hierarchies in ontologies as trees. We place special emphasis on topology understanding when designing the tool. We drew inspiration from treemaps, but made substantial changes in the representation and layout. Most notably, the spacefillingness of treemap is relaxed in exchange for visual clarity. We outline the problem space of visualizing ontology hierarchies, note the requirements that go into the design of the tool, and discuss the interface and implementation. Finally, through a controlled experiment involving tasks common to understanding ontologies, we show the benefits of our design.

1 Introduction

The vision of the Semantic Web is a meta-data rich Web where presently human-readable content will have machine-understandable semantics. The Web Ontology Language (OWL) is a W3C recommendation that allows modelers to use its expressive formalism to define various logical concepts and relations¹. A content-creator can use appropriate ontologies to, for example, annotate existing Web content. The enriched content can then be consumed by machines to assist humans in various tasks.

However, expressive ontologies can be difficult to understand. Content-creators often need to locate and inspect concepts of interest in detail to determine whether specific concepts are suitable for their use. The hierarchical structure of the concepts in an ontology can reveal a great deal about how these concepts are organized and how they are intended to be used. Effective presentation of the hierarchies can be a big win for the users.

In an OWL ontology, if we ignore `owl:Thing` as the root of the tree, and view the structure starting at the second level, an OWL ontology hierarchy can take the form of list(s), tree(s), multitrees [9], or a direct acyclic graph. One may believe that cycles of subclasses can occur. However, since the classes define sets, a cycle of subsets indicate that all classes in the cycle are equivalent. In OWL ontology editors such as Protégé [17] or SWOOP [14] the class trees are shown as standard tree widgets. Although the

¹ In this paper, we will use the term concept and class interchangeably.

widget is adequate for browsing node labels, it gives no additional information on how bushy or how deep a subtree is without further expanding it.

We present CropCircles, a technique to enhance user's ability to view the class structure at a glance. CropCircles is a tree visualizer, and like a treemap [13], CropCircles uses containment to represent the parent-child relationship. However, CropCircles sacrifices the space-fillingness for better visual clarity, enhancing understanding of the topology. This paper presents the design goals of CropCircles, the interface, and a validation of our design through a controlled experiment with treemap and SpaceTree.

2 Related Work

There are a number of ontology visualization tools available. Most of them are derivatives of tree visualizers. There are two major types of representations of trees. One is the traditional node-link diagram. The other is using geometric containment. Trees represented by node-link diagrams typically suffer from inefficient use of space. The root of the tree is usually situated where there is a lot of unused space. On the other hand, the nodes in the deep part of the tree have little room among themselves.

To remedy the inefficient use of space, hyperbolic tree viewer [15] places the hierarchy on a hyperbolic plane, and then maps the plane onto a circular region. User's focus on the tree will be given more space, accentuating the structures around it. The layout of the tree smoothly animates as the user clicks and drags at different parts of the tree. OntoRama [8] uses a hyperbolic approach to view RDF graphs. OntoRama can visualize RDF serialization of an OWL ontology, which is more verbose and consequently makes it more difficult to understand the hierarchy. One problem with the hyperbolic representation is that the constant relayout makes it difficult to maintain a mental map of where the nodes are or what the structure is.

SpaceTree [18] is a tree browser combined with a rich set of interactions to help users explore the tree. The dynamic rescaling of branches to fit the available screen space minimizes user interaction. Preview icons and miniature trees are used to give users a sense of the depth, breadth, and size of the subtrees. The smooth 3-stage animation to expand/contract subtrees help keeping the context without overwhelming users. Though it is possible to view 2 subtrees simultaneously using SpaceTree, it requires some careful user interaction. OntoTrack [16] builds on SpaceTree to browse and edit the ontology. It augments SpaceTree to use cross links to represent multiple inheritance. The implementation of the cross links, however, is awkward. Sharp bends of the links occur. Link occlusion by node labels often arise, and no optimization is done to minimize edge crossings.

Instead of using edges to represent the parent-child relationship in trees, a second type of tree representation uses geometric containment. Treemap [13] is a spacefilling representation of a tree using nested rectangles. The leaf nodes can use color and size to indicate their associated attributes. Labels for the nodes are displayed in place when there is enough room. The original treemap uses the slice-and-dice algorithm [13], which often produces elongated rectangles that are difficult to see and interact with. Squarified treemaps [5] and ordered treemaps [4] have been proposed to explicitly maintain good aspect ratio of the rectangles. Although treemaps were first applied

to visualize directory structures, they have been widely applied to other areas, among them, stock market ², news ³, sports reporting [12], microarray analysis using hierarchies from the gene ontology [1], and digital photo management system [3]. As widely as used treemaps are, they are most effective when the main focus is to understand the attribute distributions of the leaf nodes. Topological understanding is not one of its strengths.

Jambalaya [19] uses nested rectangles to show the hierarchy of classes and instances. It has a treemap view option. Different relations among the classes and instances are represented via edges between them. Users can filter both node and edge types. The visualization typically can show only 3 levels deep without serious user intervention.

There have been attempts to use geometric shapes other than rectangles to implement treemaps. Voronoi treemaps [2] use iterative relaxation of Voronoi tessellation to compute a layout of arbitrary polygons to fill a screenspace. The approach aims to address the high aspect ratio problem in treemaps and to better delineate boundaries among polygons. Kai Wetzel created a circular treemap to visualize Linux directories ⁴. There is also recent work focusing on circle packing in directory viewing [20]. Though these algorithms are used to pack in the circles as tight as possible, nested circles can obviously not fill a space. However, this extra space makes it easier to distinguish the different levels of a tree.

3 Design and Implementation

Given an ontology, users typically want to find out whether some classes in the ontology is suitable for their use. They are interested in how many subclasses a particular class has, as these subclasses are more specific than their parents and are differentiated from their siblings. In an unknown ontology, by exploring the larger branches of the hierarchy, a user is more likely to find out what the ontology is about. Likewise, in an inferred tree, one can tell that classes that lack subclasses and are children of `owl:Thing` are often undermodeled. By comparing the structural differences between the told the inferred class hierarchies, a user can also tell whether an ontology is mostly asserted, or is intricately modeled. An effective visualization should allow users to comparatively distinguish depth, bushiness, and size of subtrees.

The subsumption hierarchy in OWL is a directed graph, and to visualize it as a graph is natural and has the advantage that we do not need to duplicate nodes that have multiple parents. However, this often creates nonplanar graphs where intersecting edges cannot be avoided. Cross links are not desirable for both aesthetic and usability issues. As a result many graph drawing approaches name minimal edge-crossing as a requirement [6] [11]. Matrix representations of graphs represent edges implicitly, avoiding messy edge crossings and occlusions. But it is difficult to, for example, find how many subclasses a concept C has. This is a natural task in a tree, however. A user only needs to explore the subtree rooted at C . In a tree structure, users can better recognize the

² <http://www.smartmoney.com/marketmap/>

³ <http://www.marumushi.com/apps/newsmap/newsmap.cfm>

⁴ <http://lip.sourceforge.net/ctreemap.html>

bushiness at a certain node and whether a branch can lead to a deep node. By imposing tree structures onto a graph, we believe this will enable users to perform these tasks better.

Treemap's ability to show multiple branches and multiple levels simultaneously is attractive. It allows users to compare depth, bushiness, and size of several subtrees at once. However, despite adjustable border size and depth filters, it is still difficult to gather topological information. In particular, treemaps emphasize on visualizing leaf node attributes. The intermediate nodes are deemphasized. In visualizing ontology hierarchies, however, intermediate nodes are as important as leaf nodes. Scanning for a node's children is also problem, as they are scattered in 2D space, and labels can be cropped or completely hidden.

Our visualization design requirements are aimed to address the problems and tasks outlined above. They are summarized below.

- **Topology Overview.** In supporting the tasks to discern size, depth, and bushiness, we aim to show multiple subtrees and multiple levels at once in a tree structure. This should allow users to better comparatively gauge the subtrees. But unlike treemaps, we sacrifice spacefillingness to increase clarity.
- **Linearity in Node Reading.** At any level of the tree, the user should be able to quickly read the labels of the children. Node-link representation of trees usually have sibling nodes arranged closely on a line or a curve. Reading and counting node labels in such situations is easy.
- **Node Duplication Detection.** Because we are imposing a graph structure onto a graph, we need to support users to detect duplications due to multiple inheritance.
- **Aesthetics.** Though not the most important requirement, we feel that a visually pleasing presentation would encourage users to look at the data and gain insights from the visualization.

3.1 Layout

In CropCircles circles represent nodes in a tree. Every child circle is nested inside its parent circle. Every circle's diameter is proportional to the size of the subtree rooted at that node. The smallest subtrees (the leaf nodes) have the minimum size of all circles. For every node, CropCircles sorts its children in descending order according to their subtree sizes, and then lays them out in that order. The sorting creates a sense of order in an otherwise unordered 2D space within a circle. The larger nodes occupy more space, showing more importance, and encourage users to explore them first.

Depending on the size distribution of the children nodes, we employ 4 different layout strategies. These layout strategies are aimed to allow users to quickly gauge how the subtree sizes are distributed. To recognize whether a node has subtrees of equal size, a single subtree, or a predominant subtree can aid users' decisions on whether to explore such node further. When there is only one single child, the child node is concentrically placed inside its parent. When there are a number of equal sized children nodes, they are laid out on a concentric circle inside the parent, uniformly distributed. When there are no dominant children (a subtree that has more than 37% of the total number of descendants the parent subtree contains), all children are laid out along the lower arc of its

parent, the largest node first. When at least one dominant child is present, smaller children are laid out on an arc, equidistant from the center of the largest child. The layout for dominant child introduces a deviation from the philosophy of the other 3 layouts. In this case, the arc the layout relies on does not depend on the parent node. It brings focus to the dominant child, and gives users a visual cue that something different is there. The four layout strategies can be seen in Figure 1⁵.

In addition to the layout strategies, every child is rotated by an amount, proportional to its size-sorted rank, with respect to the either the center of the parent, or the center of its largest child. This lessens the regularity of the layout, and the result is more visually pleasing. Because of these intricately nested circles, we name our tool CropCircles. We believe the sorting of child nodes and the regularity in layout can facilitate user's understanding of the structures where the current circle-packing approaches such as Kai Wetzel's circular treemaps and [20] are not placing emphasis on.

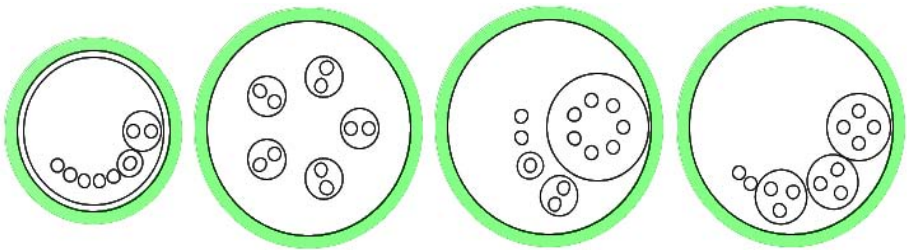


Fig. 1. From left to right, the figure shows the different layout strategies: single-child, all-children-of-equal-size, dominant-child, and no-dominant-child

3.2 Interface

The interface for CropCircles has two major components: a visualization area to the right, and a panel serving both navigation needs and detailed views to the left (see Figure 2). The visualization area shows the tree as nested circles. Users can left click on any circle to highlight that subtree in white. Each double click on a circle will pan and fit that subtree to the screen. All the zooming and panning are done in one step (no animation). Mousing over any circle will show its statistics: label of the node, depth at which it is rooted, and how large its subtree is. To effectively support multiple inheritance, when users select a circle, all other duplicated circles will be highlighted, making it easy to spot all the parents of the selected node.

For each selected node, all its immediate children are displayed on a list to the left of the visualization. Above the list, user can utilize the navigation buttons to move forward and backward to previously selected nodes. If multiple ontologies are being visualized, users can select to view any subset of them in the ontology list on top of the navigation buttons. Instead of seeing a list of immediate children, users may elect to see

⁵ We note that the colors in the screen shots have been altered to ensure readability. For unaltered images, please see <http://www.mindswap.org/~tw7/work/iswc2006/CropCircles/>

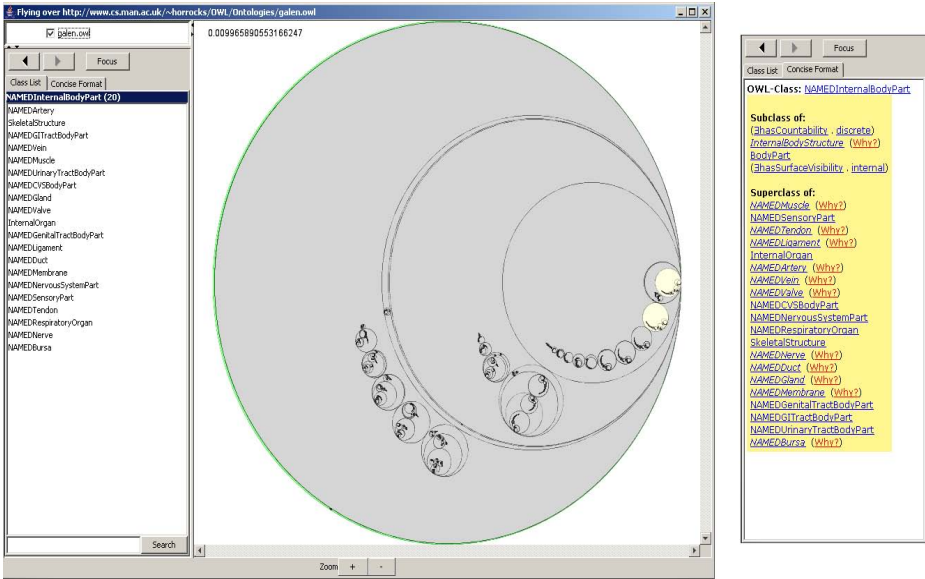


Fig. 2. This figure on the left shows the CropCircles interface. The visualization on shows the *inferred* class tree of galen. Note that the class "NAMEDInternalBodyPart" is multiply inherited in two places (hence the highlights). The left panel shows the children of the class. Alternatively, user can select the Concise Format tab to see the definition and inferred facts about the class, as shown in the right figure.

its definitions by click on the tab "Concise Format". Finally, there is a search box for name lookup. The current implementation of CropCircles uses the JUNG ⁶ framework, and is downloadable as part of the open source OWL ontology editor SWOOP ⁷.

4 Empirical Evaluation

4.1 Choice of Tools

In the controlled experimnt, we chose to compare the following 3 tree visualizers: CropCircles, treemap (Treemap 4.1 ⁸), and SpaceTree (SpaceTree 1.6 ⁹). Although treemaps are best when used to visualize node attributes at the leaf level, because much of our design decision was derived from treemap, we want to show that CropCircles is an improvement over treemaps on topological tasks. On the other hand, SpaceTree has been shown to be effective in conveying the structure of trees well in several tasks, though not without its own weaknesses [18]. We show that CropCircles is effective in conveying topological information and addresses the weaknesses of the other two

⁶ <http://jung.sourceforge.net/>

⁷ <http://www.mindswap.org/2004/SWOOP/>

⁸ from <http://www.cs.umd.edu/hcil/treemap/>

⁹ from <http://www.cs.umd.edu/hcil/spacetreel/>

tools. We used the default settings on Treemaps in our experiment. SpaceTree uses a left-to-right layout and triangle and miniature trees as visual cues. Figure 3 shows visualizations of the experimental data using the three tools.

4.2 Choice of Tasks

For each tool, we ask users to perform the following tasks.

- **Find Unknown Node.** Users are asked to find a node in the hierarchy. They are not allowed to use search boxes. Users must rely on the visual representation of the trees and labels to find the node, and have up to 2 minutes to complete the task. If they are unable to finish the task within the time limit, the experiment administrator shows the user the steps to find the node. Users are asked to perform this task twice, each time with a different target node.
- **Return to Previously Visited Node.** Users are asked to locate a node that they found in a previous node-finding task. Users are asked to click on the node to show that they have found it. They can rely on their memory of the location of the node or any navigational interfaces the tool supports. Users have up to 2 minutes to complete the task.
- **Comparison of Subtrees.** Users are asked to compare and contrast two subtrees. The experiment administrator brings the tool to a state where both subtrees are visible. Users are then free to explore the subtrees to state any structural similarities and differences of the two subtrees. Users are told to ignore label similarities, but are welcome to use them as references.
- **Find the Bushiest Child Node.** Given a node, users are asked to identify which one of its child nodes has the most immediate children. Users have up to 2 minutes to complete the task.
- **Find the Largest Subtree.** Given a node, users are asked to identify which one of its child nodes has the most descendants. Users have up to 2 minutes to complete the task. The node given to the participants has 18 immediate children, and total of 207 descendants.
- **Find a Deepest Node.** Given a subtree, users are asked to find a node that resides at the deepest level they can find. A time limit of 3 minutes is enforced.
- **Find 3 Nodes with at Least 10 Children.** Users are instructed to find 3 nodes that have at least 10 immediate descendants. Time limit is 2 minutes.
- **Find 3 Top-level Nodes that Root a subtree of Depth of at Least 5.** Users are asked to find 3 top level nodes (children of `OWL:Thing`) that root a subtree with depth of at least 5. Two minute limit is enforced.

Node-finding is an elementary task for any tree visualizer. When ontology users wish to use an ontology that is potentially suitable for their purposes, they must locate the class(es) they are interested in in order to examine if the modeling of these classes are compatible with the users' intended usage.

Ontology browsing often requires successive browsing of semantically related concepts. However, these related concepts often are not closely related in the hierarchy. That is, these concepts may not have an ancestor-descendent or even a sibling relationship.

One concept may be related to multiple concepts semantically. Users may need to adopt a breadth-first browsing pattern on the semantic relations to gain understanding of the specific concept semantically. A tool that allows users to quickly return to previously visited nodes would be favored.

Structural similarities are not uncommon in ontologies. The obvious case are the concepts that have multiple parents. These subtrees would be duplicated within an ontology. However, when an ontology imports another, and builds on top of the imported ontology, subtrees of different parents may no longer be the same. To be able to visually recognize similar trees is a plus. For example, the Federal Enterprise Architecture Reference Model (FEARMO) ontology¹⁰ makes heavy reuse of imported concepts. By inspecting the structure alone and knowing where subtrees are reused, one can quickly grasp the modeling patterns.

The last five tasks have to do with topology of the tree. Tree topology in an ontology conveys information about where in the ontology the most well-defined parts are. The number of immediate children of a node indicate how fine-grained this particular concept is being modeled. The depth of a subtree indicates how specific a particular concept is modeled. Of course, the size of the subtree is a reflection of the above two measures.

4.3 Choice of Data

We use an older version of NASA SWEET JPL ontologies as our data¹¹. Since the ontologies import one another, we stitched them together into a single file without changing the semantics. There are a total of 1537 defined classes. Adding the duplicate subtrees due to multiple inheritance creates a tree of 2104 nodes. We use this told tree for the first 3 tasks we described above. We then turn on an OWL reasoner to obtain the inferred tree, which contains 2007 total nodes. These two trees have sufficiently different topology. The told tree has a maximum depth of 11, average depth of leaf nodes 4.2, maximum branching factor 154, average branching factor of non-leaf nodes 3.9, and 103 nodes that have multiple inheritance. The inferred tree has the following, respectively, statistics: 12, 5.1, 74, 3.7, 125. We use the inferred tree to carry out the experiments on the topological tasks to prevent the effect of user learning the topology of the tree performing the first 3 tasks.

To mitigate users' possible prior familiarity with the ontology and the domain knowledge, we obfuscate the ontology by renaming the classes. The class are renamed in a pre-order traversal fashion. Given a starting integer N , the root of a subtree is given the name "CN". Then a pre-order traversal takes place to rename all its descendents recursively by incrementing N everytime a new node is encountered. We keep track of which number has been assigned to which node, so when duplicate nodes are encountered multiples in the traversal they can be assigned the same names. We create 3 pairs of the told tree and the inferred tree, every pair using a different starting N . We then cross-pair a told tree and an inferred tree so that each tree in every pair has different

¹⁰ <http://www.topquadrant.com/owl/2004/11/fea/FEA.owl>

¹¹ <http://www.mindswap.org/ontologies/debug-sweet-jpl.owl>. The most current version can be obtained via <http://sweet.jpl.nasa.gov/ontology/>

starting N . One pair is used for one tool in the experiment. We explain how the nodes are numbered prior to the experiment so users can search for nodes they have not seen before.

4.4 Experimental Setup

There are 18 subjects in our study. They are computer science graduate or undergraduate students or researchers who are familiar with tree structures. We give an overview to each subject on what they will be asked to do, and begin 3 sessions of training and experimentation phases, one session per tool. In each session, we spend up to 10 minutes training the subject on how to use the specific tool in that session. We make sure the subject understands the visual representation of the tree, whatever visual cues are available in the specific tool, ways to navigate the tree, and how to obtain vital tree topology statistics (depth of a node, size of a subtree, etc.). We show users how to use features in the tools to help them accomplish the tasks effectively. In particular, we train users to use the depth filter in treemap, and to use bookmarks in SpaceTree. After the directed instructions are done, the user is allowed to freely experiment with the tool. When a user is comfortable with the tool, or when the 10 minute time is up, we proceed to the experimental phase. After each experimental phase, we ask users to fill out a section of a survey pertaining to the tool they just used with respect to the tasks they just performed. After the experiment, users are asked to complete the survey for other feedback and background information.

The order in which the tools are presented to the users are counterbalanced to eliminate learning effects and fatigue. All experiments are done on an IBM T41 laptop with 1.4GHz CPU 1.28GB of RAM running Windows XP. Each tool occupies 1240x740 pixels. The entire experiment does not exceed 75 minutes.

5 Results

We analyze each task separately. For each continuous dependent variable (e.g. time), we use a repeated measures one-way ANOVA. We check for the sphericity condition and then perform a modified F test in the ANOVA as suggested by [10]. This means that we first conduct an unmodified univariate F test, and if the test is not significant, we accept the null hypothesis that the means obtained across the three tools are not different. If this first test is significant, we then lower the degrees of freedom and perform a conservative F test, which relies on, in our case, $F_{1,17}$ distribution. Tukey's HSD method is used for multiple comparisons when sphericity is not grossly violated. Otherwise we use Bonferroni correction on subsequent pairwise two-tailed t-tests. We use $p = 0.05$ for the ANOVA and the post hoc procedures (note that the Bonferroni correction will lower p to ensure the overall significance is 0.05). For binomial variables (e.g. success/failure), we use Cochran-Mantel-Haenszel general association statistic (which follows χ^2 distribution) to test whether the response profiles of the three tools are different [7]. We then use Bonferroni correction on the pairwise comparisons. Here we also use $p = 0.05$ for the CMH test and the pairwise comparisons. In the following sections, we present the experimental results, observations, and offers possible explanation to the observed

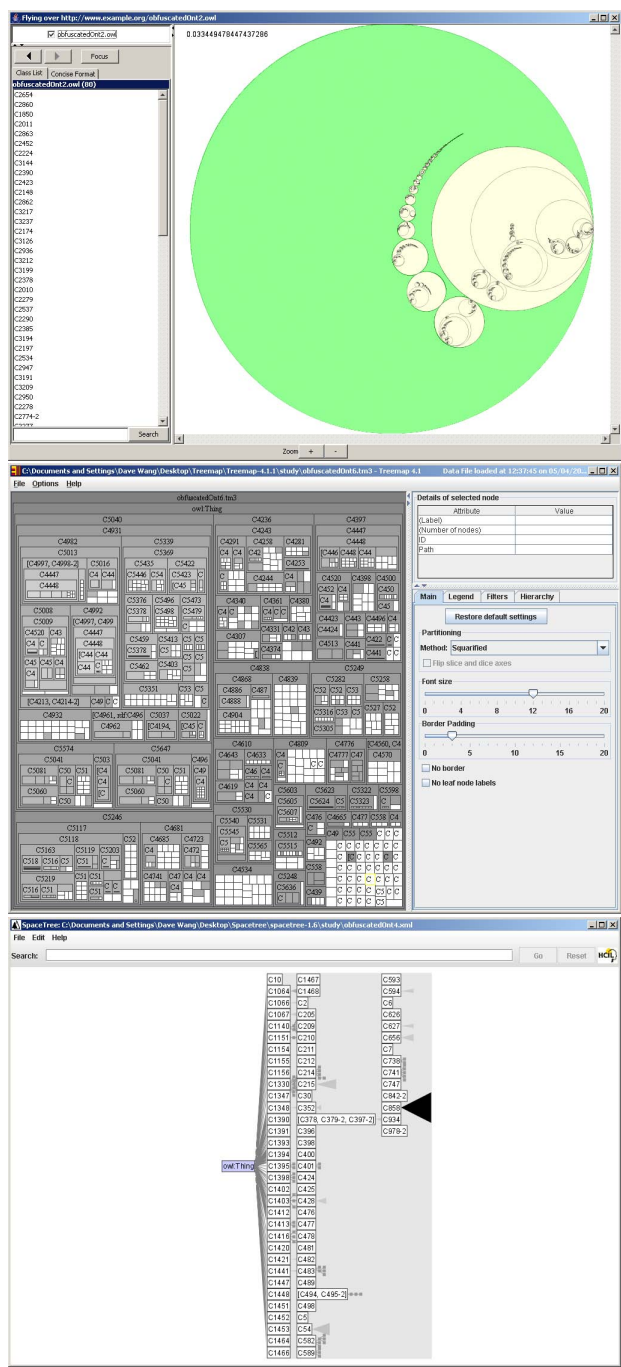


Fig. 3. The initial view of the inferred tree in the study by the three tools (top-down): CropCircles, Treemap 4.11, and SpaceTree 1.6

results. The results are summarized in Table 1. Unless otherwise specified, units used on performance is time in seconds.

Table 1. Results of the experiments. Each cell shows the recorded mean and the standard deviation of the dependent variable for a particular task and a particular tool. The statistic column shows the relevant statistics used and the level of significance obtained (if the test is significant). * denotes $p < 0.01$, + denotes $p < 0.05$. The last column shows only the statistically significant findings of the three tools: (C)ropCircles, (T)reemap, (S)paceTree. $X > Y$ indicates that X outperforms Y with statistical significance.

Task	C		T		S		Statistic	significance
	mean	SD	mean	SD	mean	SD		
Node Finding 1	87.6	34.24	77.87	37.9	30.40	13.58	$F_{1,17} = 20.53^*$	$S > C, S > T$
Node Finding 2	62.31	34.0	63.42	31.6	28.91	13.13	$F_{1,17} = 15.85^*$	$S > C, S > T$
Return to Visited Node	19.94	7.64	59.75	41.23	17.44	5.34	$F_{1,17} = 15.86^*$	$C > T, S > T$
Subtree Compare (succ. rate)	1.0	0.0	0.78	0.43	0.83	0.38	$\chi^2_2 = 5.2$	none
Bushiest Child	27.42	19.04	55.45	31.70	12.15	7.88	$F_{1,17} = 19.08^*$	$C > T, S > T$
Largest Subtree	26.09	16.25	39.23	17.88	34.66	17.54	$F_{2,34} = 2.97$	none
A Deepest Node (error rate)	0.22	0.43	0.67	0.49	0.33	0.49	$\chi^2_2 = 6.93^+$	$C > T$
3 Nodes with ≥ 10 Children	19.56	6.14	26.14	23.39	53.59	27.27	$F_{1,17} = 15.26^*$	$C > S, T > S$
3 Subtrees of Depth ≥ 5	47.90	20.4	54.09	27.81	50.84	11.86	$F_{2,34} = 0.40$	none

5.1 Navigational Tasks

- **(First Time Node-Finding)** SpaceTree performed significantly better than CropCircles and treemap. However, there was no statistically significant difference between CropCircles and Treemap. Seven participants used the depth slider in Treemap to help reduce visual clutter. Participants also used the fact that they can read labels at multiple levels of tree simultaneously to their advantage in Treemap. Participants had problems finding nodes in CropCircles. The list of size-sorted labels is difficult to digest, particularly at the root level, where there are 154 branches to explore.
- **(Return to Visited Node)** In returning to a previously visited node, both CropCircles and SpaceTree outperformed Treemap with statistical significance. There was no significant difference between CropCircles and SpaceTree. Treemap's representation does not seem to help users much in this task. In fact, the relayout to achieve space-fillingness at each zoom disorients users even though they have already done the same traversal once before. Though CropCircle users can use history to navigate backwards, only 3 participants used it. Most participants used their memory on the traversal process on the class list to accomplish this task. Participants associated well with the steps they had taken to find the node using SpaceTree. Many remembered where to look at each stage of tree expansion.

5.2 Topology Recognition Tasks

- **(Subtree Comparison)** Although all participants were successful in making the observation using CropCircles, and some portions of participants failed in Treemap and SpaceTree, the differences among the tools are not statistically significant.

- **(Finding the Child Node that has the Most Immediate Children)** CropCircles and SpaceTree allowed users to complete this task significantly faster than Treemap. But there is no statistical significance between CropCircles and SpaceTree. When the target node was expanded in SpaceTree, it fully expanded both its child and its grand child level, but not its great-grand child level. This is exactly the right amount of information users needed to complete the task. The children nodes are presented in a linear list, making it easy to count. Many participants were observed to use the depth slider in treemap to filter out unnecessary nodes to quickly obtain the answer.
- **(Finding the Largest Subtree)** There was no statistical significance among the three tools. This was a surprising result. We observed that although participants are told that the nodes were sorted by size in CropCircles, users would spend time to verify the sizes as if they do not trust the visual representation. Similar situation is observed in SpaceTree. Users moused over all children to read the size of the subtree reported in the tooltips when only the subtrees with dark preview triangles should require closer inspection.
- **(Finding a Deepest Node)** We measured how successful users were at finding a node that is at the deepest level of the given subtree. We performed analyses on the error rate. CropCircles had significantly lower error rate than treemap, but the difference between SpaceTree and CropCircles was not significant. There was also no significant difference between SpaceTree and treemap.
- **(Finding 3 Nodes with at Least 10 Immediate Descendents)** Both CropCircles and Treemap outperformed SpaceTree significantly, but there was no statistically significant difference between the two. The nodes that CropCircles users reported tend to be at the upper levels of the tree, as they took no additional zooming to see. On the contrary, all nodes reported by Treemap users are the ones that contain many leaf nodes, which are white, and are easy to see.
- **(Finding 3 Top-Level Nodes that Root Subtrees of Depth of at Least 5)** There were no statistically significant differences among the three tools.

6 Discussion

Ignoring statistically insignificant results, CropCircles performed well against Treemap and SpaceTree in topological tasks. CropCircles edged SpaceTree in finding 3 nodes with at least 10 children, and was better than Treemap in finding a deepest node and finding the bushiest child. Although there was no one task that CropCircles was better than both of the other two tools, there was also no one topology task that CropCircles performed unsatisfactorily. In this sense CropCircles is the most balanced of the three tools in topology recognition tasks. By avoiding Treemap's and SpaceTree's weaknesses, CropCircles is an appropriate visualization for class hierarchy. For example, ontology modelers who wish to visually explore where an ontology is under-modeled (characterized by subtrees that lack depth and bushiness in the inferred tree), CropCircles would be a good choice. The results also suggest that ontology hierarchy visualizers that use SpaceTree or treemap as the underlying technique should be aware of their shortcomings and address them.

On the other hand, not all of our design decisions were validated. Although listing children in a list that enables level-traversal allows users to remember the path they took to a particular visited node, the list is inadequate to support label browsing. An option to sort the labels alphabetically would have helped the users a great deal in node-finding tasks. We were also not able to show that CropCircles can outperform the other tools with statistical significance in finding the largest subtree, even though the subtrees are ranked by size.

Our participants gave us valuable feedbacks on how to improve CropCircles in our post experimental survey. Many mentioned better context support when details are focused. Several users suggested a more tightly integrated history with the visualization. Almost all participants commented on the lack of support to sort node labels alphabetically. Information density in CropCircles is a concern, and several users have mentioned the desire to see the space utilized better. These comments and our experimental results are observed, and will be the main focus in the next step of our iterative design process.

7 Conclusions

We describe CropCircles and our requirements in designing a tool to visualize the topology of OWL class hierarchy. While our design exploited several useful principles, not all design decisions are helpful in completing the tasks in the experiments. However, we are able to show that in topological tasks, CropCircles's performance is comparable to strengths of the two other tools, and is an improvement over their known weaknesses. This result makes CropCircles an attractive alternative in viewing class hierarchies in OWL.

Acknowledgments

This work was supported in part by grants from Fujitsu, Lockheed Martin, NTT Corp., Kevric Corp., SAIC, the National Science Foundation, the National Geospatial Intelligence Agency, DARPA, US Army Research Laboratory, and NIST. Special thanks to Jennifer Golbeck for her helpful comments and suggestions.

References

1. Eric H Baehrecke, Niem Dang, Ketan Babaria, and Ben Shneiderman. Visualization and analysis of microarray and gene ontology data with treemaps. *BMC Bioinformatics*, 84(5), 2004.
2. Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the IEEE Symposium on Information Visualization*, 2005.
3. Benjamin B. Bederson. Quantum treemaps and bubblemaps for a zoomable image browser. In *Proceedings of User Interface Systems and Technology*, pages 71–80, 2001.
4. Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Transactions on Graphics*, 21(4):833–854, 2002.

5. Mark Bruls, Kees Huizing, and Jarke J. van Wijk. Squarified treemaps. *Proc. IEEE Symposium on Information Visualization '99*, pages 284–291, 2000.
6. Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Tran. on Graphics*, 15:301–331, 1996.
7. Charles S. Davis. *Statistical Methods for the Analysis of Repeated Measurements*. Springer, 2002.
8. P. Eklund, N. Roberts, and S. P. Green. Ontorama: Browsing an rdf ontology using a hyperbolic-like browser. In *Proceedings of the 1st International Symposium on CyberWorlds (CW2002)*, pages 405–411, 2002.
9. G. W. Furnas and J.Zacks. Multitrees: Enriching and reusing hierarchical structure. *Proceedings of ACM CHI 1994 Conference on Human Factors in Computing Systems*, 1994.
10. S. W. Greenhouse and S. Geisser. On methods in the analysis of profile data. *Psychometrika*, 29:95–112, 1959.
11. David Harel and Meir Sardas. Randomized graph drawing with heavy-duty preprocessing. *Journal of Visual Language and Computing*, 6:233–253, 1995.
12. Liquin Jin and David C. Banks. Tennisviewer: A browser for competition trees. *IEEE Computer Graphics and Applications*, 17(4):63–65, 1997.
13. Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd International IEEE Visualization Conference*, pages 284–291, 1991.
14. A. Kalyanpur, B. Parsia, and J. Hendler. A tool for working with web ontologies. *Int. J. on Semantic Web and Info. Syst.*, 1(1), 2004.
15. J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. *Conference Proceedings on Human factors in computing systems*, pages 401–408, 1995.
16. Thorsten Liebig and Olaf Noppens. OntoTrack: Combining browsing and editing with reasoning and explaining for OWL Lite ontologies. In *Proceedings of the 3rd International International Semantic Web Conference*, 2004.
17. Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Ferguson, and Mark A. Musen. Creating semantic web content with protégé-2000. *IEEE Intelligent Systems*, pages 60–71, 2000.
18. Catherine Plaisant, Jesse Grosjean, and Benjamin B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Proceedings of IEEE Symposium on Information Visualization*, pages 57–64, 2002.
19. M.-A. D. Storey, M. A. Musen, J. Silva, C. Best, N. Ernst, R. Ferguson, and N. F. Noy. Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé. *Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*, 2001.
20. Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. Visualization of large hierarchical data by circle packing. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*, pages 517–520, 2006.