CAR-TR-772 March 1995
CS-TR-3474
ISR-TR-95-53

Browsing Hierarchical Data with Multi-Level Dynamic Queries and Pruning

Harsha P. Kumar, Catherine Plaisant and Ben Shneiderman*

Human-Computer Interaction Laboratory
Department of Computer Science
Institute for Systems Research*
University of Maryland, College Park, MD 20742-3255
harsha@src.umd.edu, plaisant@cs.umd.edu, ben@cs.umd.edu

Abstract

Users often must browse hierarchies with thousands of nodes in search of those that best match their information needs. The *PDQ Tree-browser* (**P**runing with **D**ynamic **Q**ueries) visualization tool was specified, designed and developed for this purpose. This tool presents trees in two tightly-coupled views, one a detailed view and the other an overview. Users can use dynamic queries, a method for rapidly filtering data, to filter nodes at each level of the tree. The dynamic query panels are user-customizable. Subtrees of unselected nodes are pruned out, leading to compact views of relevant nodes. Usability testing of the PDQ Tree-browser, done with 8 subjects, helped assess strengths and identify possible improvements. The PDQ Tree-browser was used in Network Management (600 nodes) and UniversityFinder (1100 nodes) applications. A controlled experiment, with 24 subjects, showed that pruning significantly improved performance speed and subjective user satisfaction. Future research directions are suggested.

please address correspondence to Ben Shneiderman

To appear in *International Journal of Human-Computer Studies*

1 Introduction

Decisions are an integral part of human life. Whether deciding what movie to see or choosing which universities to apply to, people are constantly faced with decisions. Many everyday decisions and in engineering applications require the selection of one or a few elements from many, possibly thousands of elements. In such cases, users often try to make a ``good'' choice by deciding first what they do *not* want, i.e. they first try to reduce the data set to a smaller, more manageable size. After some iterations, it is easier to make the final selection(s) from the reduced data set. This *iterative refinement* or *progressive querying* of data sets is sometimes known as hierarchical decision-making.

A hierarchical data set (HDS) (also called a ``tree'') organizes data points into a hierarchy. HDSs are common, e.g. sales & budget data, catalogs of products, library indices, computer file systems etc. A HDS can be filtered (queried and reduced) effectively using a hierarchical-decision making process because the data is inherently hierarchical.

The original motivation for this work was our research on user interfaces for network management (Kumar, Plaisant, Teittinen, & Shneiderman, 1994). While working on a user interface for satellite network configuration, we were faced with the task of choosing one leaf node from a large tree of thousands of nodes. Further, the task was such that the number of interesting leaf nodes could be reduced drastically based on selection criteria at various levels in the tree. This problem prompted us to design and implement the PDQ (**P**runing with **D**ynamic **Q**ueries) Tree-browser.

2 PDQ Tree-browser Requirements

Based on our task analyses in the network management scenario and other tree-browsing applications, we specified the requirements for the PDQ Tree-browser visualization tool:

- Browse the entire tree and view it at different levels.
- Query nodes at all levels on the basis of attribute values. Querying mechanism should be easy, rapid, yet powerful.
- Hide uninteresting nodes and branches rapidly, and thus reduce the data set progressively. Iterate easily by revealing hidden nodes / branches.

3 Previous Work

Information search is a vast topic (Marchionini, 1995), but previous work on hierarchical data is more limited. Early work emphasized aesthetic and easy-to-read tree and graph layouts (Battista, Eades, Tamassia, & Tollis, 1989), and browsing / exploring / searching trees and graphs, but none of them provide a good solution to the attributes-based querying and browsing problem. Many visual browsers address 2D-browsing, and some allow for manual pruning of uninteresting subtrees / subgraphs. Commonly used tree visualizations include 2D node-link diagrams, space-filling treemaps, 3D node-link diagrams and tables-of-contents or outliners.

An advantage of node-link visualizations of trees is that they are a familiar mapping of structured relationships and therefore easy to understand. They can also display attributes of links by color or size if required. However, node-link diagrams make inefficient use of screen space, and even trees of medium size need multiple screens to be completely displayed. This necessitates scrolling of the diagram and global context is lost, since only a part of the diagram is visible at any given time.

Beard & Walker (1990) used a *map window* - a miniature of the entire information space with a wire-frame box to aid users in remembering their location. The map window is better known as an *overview* , the entire information space shown in full size is the *Detailed View* and the wire-frame box is the *field-of-view* or the *panner* (Plaisant, Carr, & Shneiderman, 1995). The field-of-view can be dragged around in the overview to pan the detailed view. Similarly, scrolling the detailed view updates the position of the field-of-view in the overview. Hence, the overview and the detailed view are said to be *Tightly-coupled*. Beard & Walker found that an overview significantly improves user performance, and the pan technique and the zoom and pan technique were significantly faster than the scroll technique.

Plaisant *et al.* provide a taxonomy and guidelines for image-browsers. This work attempts to standardize some of the terms being used by researchers today, e.g. Detail View, Field-of-view, etc. Different kinds of browsers such as ``Detail only'', ``One window with zoom and replace'', and ``Tiled multilevel browser'' are shown. The authors identify 5 classes of tasks that are accomplished with image browsers, e.g. open ended exploration, navigation, monitoring, etc.

The treemap visualization of tree structures uses a two-dimensional (2D) space-filling approach in which each node is a rectangle whose area is proportional to some attribute such as node size (Shneiderman, 1992). On the other hand, the treemap algorithm utilizes 100 % of the designated space. Sections of the hierarchy containing more important information can be allocated more display space while portions of the hierarchy which are less important to the specific task at hand can be allocated less space. Treemaps have been used for file management, network management (Kumar *et al.*, 1994), budgets, sports data, etc.

Robertson, Mackinlay, & Card (1991) and Chignell, Zubrec, & Poblete (1993) used 3D node-link diagrams in order to visualize tree structures. Robertson *et al.*, in the Information Visualizer project at Xerox PARC, developed a tool called Cone Trees that allows for animation of 3D trees. They contend that interactive animation can effectively shift cognitive processing load to the perceptual system. They describe *gardening operations*' (Robertson *et al.*, 1991) where the user can manually prune and grow the view of the tree. Prune and grow operations are done either by menu or by gestures directed at a node. While these gardening operations do help in managing and understanding large, complex hierarchies, this manual mechanism of pruning or growing one subtree at a time, is clearly not sufficient nor very effective in specifying complex searches spanning several levels of the tree.

Chignell *et al.* (1993) built the Info-TV tool, which allows two styles of pruning:

- The sub-branch(es) for which the chosen node is the root can be removed from the screen
- The nodes and labels are removed, but the links remain

The authors however, do not describe how the nodes whose sub-trees are to be pruned are specified by the user. It is assumed that the user makes these specifications manually by selecting these nodes.

Visualizations have also been used traditionally to browse of graph structures. Examples of graph visualizations include hypertext graphs, finite-state diagrams, flow-charts, parse-trees, pert-charts, dataflow diagrams, wiring diagrams and hierarchical decompositions.

According to Henry (1992), and Henry and Hudson (1991), the ``best layout" depends on the user's current region of interest. Consequently, a single layout algorithm cannot always produce the best results. Thus, the *ability of users to customize* the layout to meet their current needs and interests is essential. Therefore, users must be provided with *interactive tools* to *iteratively* dissect large graphs into manageable pieces. Henry and Hudson describe manual selection in which users select nodes and edges using simple direct manipulation techniques, and algorithmic selection in which users apply an algorithm to the graph. But by classifying manual selection into *only* individual selection or marquee selection, the authors are greatly restricting the range of interesting selection subsets that can be specified by the user. They conclude by identifying two primary future directions: using domain specific graph semantics to guide the layout and the selection, and creating a methodology that can be used to build *an interactive system for non-programmers to specify selection and layout algorithms*. Our work extends their strategies.

Gedye (1988) built a system that presents users with a list of all the objects. They can choose an object, and then upon choosing one particular relationship, a directed acyclic graph, tree, or an equivalence set is created in a new window. So, the tangled web that would have resulted by showing all the relationships is eliminated by using one window per relationship. Noting that both zooming and panning were inadequate for arbitrary graph structures, the authors implemented pruning in order to assist in the browsing. A subgraph is selected that contains few enough nodes to comfortably fit in the window, and this subgraph is displayed in its entirety. The subgraph obtained by the pruning procedure is called the "display graph".

Schaffer, Zuo, Bartum, Dill, Dubs, Greenberg, & Roseman (1996) found that users performed better using fisheye views of hierarchically-clustered graphs than using full-zoom views. Hollands, Carey, Matthews, & McCann (1989), however found users getting somewhat disoriented while using fisheye views for complex tasks. None of the fisheye view implementations described above allow for attributes-based specification of the foci of interest.

The review of the literature pertaining to tree and graph browsers did not provide a good answer to the problem of specifying selection subsets on large connected data sets like trees and graphs. Those that do allow for uninteresting nodes / subtrees / subgraphs to be pruned out, require users to make subset specifications manually. We believe that Dynamic Queries can be used effectively for this purpose. Dynamic Queries describes the interactive user control of visual query parameters that generates a rapid (100 msec update) animated visual display of database search results. Dynamic queries are an application of the direct manipulation principles in the database environment. They depend on presenting a visual overview, powerful filtering tools, continuous visual display of information, pointing rather than typing, and rapid, incremental, and reversible control of the query (Shneiderman, 1992). Dynamic queries have been applied to browser databases of houses, movies, chemical tables of elements etc. These concepts of dynamic querying and tight-coupling are similar to those of Focusing and Linking (Buja, McDonald, Michalak, & Stuetzle, 1991).

4 PDQ Tree-browser design

4.1 Theory

Our PDQ Tree-browser design consists of the following features:

- Two tightly-coupled node-link views of the tree (Overview and Detailed View)
- Dynamic Query Environment for users to customize their dynamic query panels
- Dynamic Queries at different levels of the tree
- Pruning of subtrees of uninteresting nodes to get more compact views

4.1.1 Dynamic Queries on Hierarchical Data Sets: Dynamic queries have been applied to data sets consisting of independent data points (Ahlberg and Shneiderman, 1994). In such cases, whether a data point satisfies a given query or not does not affect the outcome for other data points. This is because there are no interrelationships between the data points. Thus, the data set can be thought of as ``flat". Queries are merely queries on the attributes of individual data points. For example, in the FilmFinder, each movie is an independent data point. Similarly, in the HomeFinder (Shneiderman, 1994), each house is an independent data point.

In a "non-flat" data set, on the other hand, there are predefined interrelationships between data points. For example, in a HDS, some nodes are related to some others by the parent-child relationship (Kumar *et al.*, 1994). Therefore, whether a node matches a given query or not might affect some other nodes. Specifically, while searching a hierarchical data set in a top-down manner (i.e. parent first), it makes sense to prune out all descendant nodes of nodes that do not match the query. For example, if users are looking for departments in universities with low tuition, it makes sense to eliminate those departments whose universities have high tuition. When criteria (like low tuition, high average SAT scores, high placement indices etc.) exist at all or most levels in the hierarchy, *stepwise refinement of the query* can be done to progressively reduce the initial (large) data set into a smaller set, from which good choices may be made.

4.1.2 Dynamic Query Environments: The HomeFinder and the FilmFinder are examples of systems that provide hard-coded graphical widgets for the user to manipulate in order to dynamically update the visual display. If the user wanted to find homes that were within 2 miles of any hospital, the current HomeFinder interface would have to be reprogrammed.

Therefore, a *Dynamic Query Environment* , first implemented in a browser for the National Center for Health Statistics (Shneiderman, 1994), should allow users to customize dynamic query control panels based on current interests. Users should be able to select what combination of attributes they wish to query on, and have the appropriate widgets created, at run-time. The method of selecting the attributes and creating widgets should be easy, and also allow for modifications / backtracking. Having the interface not be application-dependent would have the added advantage of reusability across applications.

4.2 Description of the Interface

The PDQ Tree-browser interface consists of two main parts (Figure 1):

- **Data Display**: The tree structure is visualized in two tightly-coupled views, a detailed view (on the right) and an overview (on the left). If the PDQ Tree-browser window is resized by the user, the field-of-view shape and size is updated automatically.
- **Dynamic Query Panel**: This panel (below the Data Display) consists of two parts, the Attributes List on the left and the Widgets Panel on the right. Initially, there are no query widgets in the Widgets Panel. Users may select (by dragging-and-dropping) up to three attributes from the Attributes List for each level in the hierarchy (except the root level). This causes an appropriate widget (range-slider for numerical attributes and menu for textual attributes) to be created and initialized. Queries (on up to 3 attributes) at each level are AND-ed together. Users can replace an existing widget with another by dropping a new attribute name over the original widget's attribute name. Existing widgets can be deleted by dropping ``No Query'' onto the corresponding attribute name.

The Treeview and Range-slider widgets of the University of Maryland Widget Library ™ (Carr, Jog, Kumar, Teittinen, & Ahlberg, 1994) were used in the PDQ Tree-browser implementation.

If users manipulate a widget at the current lowest level displayed, the nodes matching the query at that level are colored yellow, otherwise they are gray. These updates of the data display are real-time (within 100 msec of updates to the control widgets) in accordance with the principle of dynamic queries. Buffering was done in order to make the updates as smooth and flicker-free as possible.

If users manipulate a widget at a level other than the current lowest level, the tree visualization first ``jumps" to that level, i.e. the level of the widget is made the tree's current lowest level. This is done so that the structure of the tree during direct manipulation of the widgets remains constant and only the colors of the nodes change. Then the nodes at the new lowest level are updated (by coloring yellow or gray) in real-time to show whether they match the query or not.

The tree structure changes *only* when the current lowest level of the tree is changed, which can be accomplished by users in three ways:

- By clicking on the corresponding level button (i.e. the buttons labeled ``Network', ``DPC", ``LIM" and ``Port") just below the data display (Figure 1).
- By manipulating a widget at any level other than the current lowest level, as explained above.
- By clicking on the + or - buttons to either increase or decrease the levels displayed.

When the current lowest level is changed so as to show more levels, *pruning* of the tree is done so as to eliminate subtrees of nodes that do not match the query at their own levels. For example, if the user manipulates the range slider for Network ID such that Nets 1 and 2 do not match the query (Figure 2), and then increases the lowest level displayed by 1, the children of nets 1 and 2 are not shown, while children of other nets are shown (Figure 3). Nets 1 and 2 now appear in gray, while all the other nets appear in orange, simply to show that those nodes did match the query at their own level. This feedback enables users to go back and change queries at higher levels, and thus iteratively refine the selection subset.

As explained above, nodes at the current lowest level are colored either yellow or gray, while nodes at higher levels are colored either orange or gray. Also, the level button corresponding to the current lowest level is colored yellow so as to focus the attention of the user to that level. Buttons at other levels are colored gray.

When the current lowest level is changed so as to show fewer levels, the tree is simply ``folded" back to the new lowest level, and then the nodes at that level are colored either yellow or gray based on the query at that level. Thus, users can easily jump back and forth between levels in order to fine-tune their search.

The Attributes List on the left shows only the attributes corresponding to the current lowest level. Users can access names of attributes at other levels by choosing the appropriate level name (e.g. Network, DPC, etc.) from the attributes menu just above the Attributes List (Figure 1).

There are four feedback indicators, one corresponding to each level (other than the root level), that are updated in order to show the number and proportion of nodes that currently match the query (i.e. number of ``hits"). The proportion of the feedback indicator that is colored yellow corresponds to the proportion of hits. This proportion is a percentage of the total number of nodes at that level, not of the number of nodes at that level currently displayed. The actual number of hits is also displayed at the top of each feedback indicator. Proportion indicators are displayed from level 1 down to the current lowest level only , those for deeper levels are hidden (grayed out). This is because calculating the number of nodes that *would* match the query at deeper levels is computationally intensive and would slow down the dynamic queries on nodes at the current level. At any time users can select the button to hide the grayed out nodes.

To summarize, the PDQ Tree-browser is a visualization tool for hierarchical data that makes use of dynamic queries and pruning. The PDQ Tree-browser uses two coordinated or tightly-coupled views of the same tree, one a detailed view and the other an overview. The user can select up to 3 attributes (numerical or textual) for dynamically querying nodes at each level in the hierarchy. Subtrees of nodes that do not match the query at their own level are pruned out of the visualization. Thus, one can reduce a large data set (with thousands of nodes) to a much smaller set, from which good selections can be made.

4.3 Example Application: The UniversityFinder

We applied the PDQ Tree-browser to 2 applications, Network Management and the UniversityFinder. The latter is described here. The database organizes universities hierarchically; we have regions of the world, followed by states, then universities, and finally, departments. The UniversityFinder demo is available on videotape (Kumar, 1995).

Let us say that I am a high-school senior looking for universities that best match my needs. The PDQ Tree-browser initially shows all the regions in the world. I ask to see the entire tree to get an idea of the size of the database. The feedback indicators show that there are 740 departments in 286 universities in 62 states in 5 regions of the world (Figure 4).

I realize that panning this entire tree is not an easy task and return to the region level. Since I am only interested in regions where English is the primary language, I create a textual menu of primary languages by dragging-and-dropping the attribute Primary Language on to one of the empty slots under the Region level button. I select English from the menu and the South America turns gray, while USA, Canada, Africa and Europe remain yellow.

I now proceed to the state level. South America's subtree is not expanded. That subtree was pruned out since South America did not satisfy our criterion for primary language. However, South America is still shown in gray in order to provide context feedback.

I would really like to study in a state that is relatively safe, so I choose to query on the level of violence. I manipulate the range-slider to select only states with a violence index less than 65, leaving 37 states. I further reduce the number of interesting states by choosing only those with good traffic conditions. The number of states has now dropped to only 7 (Figure 5).

Now, I can look at the universities, but first I recapture space occupied by uninteresting states by clicking on the Hide Gray Leaves button. This gives me a more compact view that often is visible on a single screen. When I now go to the university level, only the remaining 7 states are expanded to show 25 universities.

I now reduce the number of universities by first eliminating those with high tuition and then setting the average SAT scores to closely match mine. I hide gray leaves once again and see that I am down to 7 universities: a couple in Arkansas, a couple in Ontario, etc.

Satisfied with this set of universities, I proceed to the department level to closely look at departments (Figure 6) that best match my interests and needs in terms of availability of financial aid, etc.

4.4 PDQ Tree-browser Limitations

The UniversityFinder scenario demonstrates how the PDQ Tree-browser can be applied to everyday applications, in addition to complex applications like network management. The tool itself is general and can be used for any hierarchical data set with attributes for nodes at different levels of the hierarchy. However, the current implementation does have the following limitations:

- The PDQ Tree-browser interface has been ``fine-tuned'' for a tree of depth 5. The underlying tree data structure and node-link widgets place no constraints on the depth of the tree, but the current interface has been customized to look best for a tree of depth 5, e.g. the levels of the tree in the detailed view align nicely with the corresponding buttons and query widgets.
- Users can select only up to 3 attributes to query on at each level, and these queries can only be ANDed together. ORs and NOTs are not supported currently.
- Since the focus of this work was not on layout, the algorithm used produces an aesthetic tidy layout, but not the most compact one. Also, the implementation does not enforce the overview to always show the entire tree (so users might have to scroll the overview).
- Users can not open subtrees of nodes by manually selecting (e.g. double-clicking) them. We believe that both manual selection and attributes-based selection are necessary in a complete system.

4.5 Possible design alternatives

There are some interesting design alternatives that deserve special mention here:

- Pruning vs. graying out: The PDQ Tree-browser prunes out subtrees of nodes that do not match the query at their own level. Another approach would have been to show the entire subtree, but with all the nodes grayed out. The possible advantage of this approach over the pruning approach would be the increased constancy in the tree structure. Expert users might experience improved productivity as they get more and more familiar with the tree structure. The disadvantage of this approach is that the tree is displayed in its entirety at any level, even when most of the nodes that take up a lot of screen space are uninteresting. This results in increased overheads of scrolling and panning and slower response times. In Section 6, we describe a controlled experiment which compared these behavior alternatives.
- Whether to update the data display when a query widget is created or replaced at a level different from the current lowest level: The PDQ Tree-browser changes levels whenever widgets at levels other than the current lowest level are manipulated. But it does not change levels when a new widget is created or an existing widget replaced, at a level different from the current lowest level. This design decision was based on the assumption that users might create a number of widgets at different levels at the same time (e.g. at the beginning), and not want the level to change each time. The disadvantage of this approach is that the data display is potentially inaccurate till the next time users visit that level.

4.6 Generalizing the design

This section attempts to generalize the PDQ Tree-browser design so as to overcome some of its limitations and make it more generally applicable to trees of varying structures and sizes. Ideas on extending the query interface to allow specification of complex boolean queries are discussed. The PDQ Tree-browser illustrated the advantages of using dynamic queries and pruning while visualizing trees as 2D node-link diagrams. The same advantages are there to be had by other tree visualizations as well, e.g. treemaps and Cone Trees. We illustrate this with examples for the treemap case.

4.6.1 Coping with varying structure and growing size: In this section, we examine some of the issues that need to be addressed in order for our design to be extensible to trees of varying structure and size.

- **Trees of arbitrary depth**: Figure 7 shows how one might extend the current PDQ Tree-browser interface to handle trees arbitrarily deep. Due to screen space constraints, it will not be possible to see all the query widgets at all levels in the hierarchy. In Figure 7, on the lower left corner, is a list of levels in the hierarchy, from which users select the current lowest level (level 2 in this case). The Attributes List then updates to show the list of attributes for level 2. The query widgets for the current lowest level and for the previous level visited (i.e. level 1 in this case) are shown in the query panel.

- **Trees of arbitrary size**: As the size of the tree to be visualized becomes larger, performance would tend to deteriorate and the browsing mechanism becomes inadequate. But there are approaches that can be taken to alleviate this problem:

  Performance issues: Our node-link layout algorithm, in the worst case (i.e. when all nodes are to be shown), requires 2 complete traversals of the tree, Increasing levels displayed requires more than 3 traversals of the entire tree, in the worst case. Dynamically querying nodes at any level is more efficient in that is only requires traversal of each node at that level and not of the entire tree. Therefore, the system is likely to slow down appreciably as the size of the tree increases, especially for operations that require re-computation of the structure and layout.

  More sophisticated data structures and algorithms would be necessary in order to minimize the performance deteriorations. For example, one might only traverse the nodes visible in the detailed view to evaluate dynamic queries. But this would mean that the overview would not be tightly-coupled with the detailed view. Algorithms for pruning could be improved so that only subtrees that have been dynamically queried since the last structure change need to be traversed. There are several interesting challenges that remain to be solved, with respect to performance issues.

  Interface issues: Ideally, the overview would always show the entire data display in miniature, even if it means that no details are visible; and the overview provides only global context. However, when the size of the tree gets huge (say 50,000 nodes), there is no way that one overview would suffice. Plaisant, Carr, & Hasegawa (1992) found that in some cases, it might even be useful to provide an intermediate view in addition to the overview and the detailed view, when the detail-to-overview zoom ratio is above 20.

  Another feature that might be useful is to allow users to restrict the nodes to be displayed *before* displaying them. For example, if the user requests to see a new level with 20,000 nodes, the system should present the user with the option of restricting this set (feedback might be provided by displaying the number of matching nodes) before displaying it.

4.6.2 Specification of general boolean queries: The current PDQ Tree-browser implementation allows users to specify an AND of queries on 0-3 attributes at each level. An improved interface would allow any number of ANDs, ORs and NOTs. But specifying complex boolean queries graphically is a challenge, as it may be difficult to interpret a set of graphical widgets, boolean operators and parentheses, even for experienced users.

The PDQ Tree-browser could accommodate complex boolean queries by using the Filter-Flow metaphor in which queries are shown as water streams flowing through sequential filters for ANDs, parallel filters for ORs, and inverted filters for NOTs (Young & Shneiderman, 1993). The decrease in the breadth of the water stream while passing through a filter represents the reduction in the data set due to the corresponding query.

4.6.3 Pruning applied to treemaps: It was in fact, a treemap of a network hub that first highlighted the need for some mechanism to hide subsets of nodes / prune subtrees that were not interesting and recapture screen space. Uninteresting leaf nodes had taken up about 40 % of the total display space.

Figure 8 shows how pruning can help treemaps recapture screen space allocated to uninteresting subtrees.

4.6.4 Dynamic queries and pruning applied to cone trees: Robertson *et al.* (1991) describe ``gardening operations'' where the user can manually prune and grow the view of the tree. Prune and grow operations are done either by menu or by gestures directed at a node. We believe that allowing users to make attributes-based subset specifications in addition to these manual subset specifications will help make Cone Trees more powerful and useful.

5 Usability Testing

In order to assess and improve the PDQ Tree-browser, usability evaluations were performed, in which 8 subjects were given training and asked to perform specified tasks using the PDQ Tree-browser. The UniversityFinder data was used in the testing.

5.1 Methods

With each subject, the following procedures were followed:

- Introduction: The experiment was introduced to the subjects by explaining the UniversityFinder scenario (Section 4.3).
- Description of features: PDQ Tree-browser features were demonstrated one by one and subjects were given the opportunity to try each one.

- Tasks: Subjects were asked to perform 7 tasks. The initial tasks focused on specific features, while later tasks were designed to evaluate the tool as a whole (See Section 5.3 for details). Subjects were encouraged to think aloud while performing the tasks. Comments and suggestions were recorded as they were made. Interesting actions and sources of confusion were also recorded.
- Subjective evaluation: Subjects were asked to rate specific features of the system (on a scale of 1 to 9), identify what they liked and disliked most about the PDQ Tree-browser and make suggestions for improvement.
- Analysis: Finally, all comments and suggestions made by subjects were compiled into one list (Section 5.4). The mean and standard deviation were computed for each rating and some graphs were plotted.

5.2 Subjects

Pilot testing was done with 2 graduate students, to try out the tasks and get preliminary reactions. The tasks were refined, a subjective evaluation questionnaire was designed, and the software was tested with 6 subjects. The subjects were familiar with tree structures and pruning, database querying, GUIs and direct manipulation widgets. The subjects ranged from graduate students and undergraduate seniors in Computer Science / Electrical Engineering to a faculty research assistant.

5.3 Tasks

The range of (7) tasks included:

Feature-based testing: Testing the specific features of the PDQ Tree-browser.

- Tight-coupling of overview and detailed view: 2D browsing of the node-link diagrams using panning, scrolling and changing levels features only.
- Dynamic Query Environments: Building and modifying dynamic query panels using drags-and-drops. Directly manipulating the widgets to produce real-time color updates of the nodes.
- Tree Dynamics: Tree pruning and associated structure and layout changes. Issues relating to getting familiar with the tree structure and dis-orientation due to structural changes. Ability to iteratively refine queries, by jumping back and forth between levels.

Task-based testing: Testing queries of varying complexity and type. This classification of queries is general and independent of this particular PDQ Tree-browser design and implementation.

- Simple ``attributes-based'' queries: For example, how many states have a level of violence index <= 60 and good traffic conditions?
- Complex associative ``structure-based'' queries: For example, how many states satisfy all the following constraints: regions with standard of living >= 50 but <= 95, states with population

density <= 72 and good traffic conditions, public universities with out-of-state full-time tuition <= $3000 per semester and average SAT scores of >= 1100 and at least one department.

5.4 Results

The subjective evaluations revealed which features were liked and which needed improvement:

5.4.1 Most liked features: When asked what was the one thing they liked the most about the PDQ Tree-browser, subjects responses included:

- ``Dynamic querying and pruning to get multiple views based on current interests."
- ``Easy to learn, convenient and straight-forward to use."
- ``The ability to locate interesting parts of large trees."
- ``Easy to create complex multi-level queries."
- ``I liked seeing the tree structure of the data, which would usually be tabular."
- ``Shows adequate information."
- ``Tightly-coupled overview and detailed view."

5.4.2 Subjective ratings: Users rated 22 aspects of the PDQ Tree-browser on scales of 1 to 9, where 1 was the worst rating and 9 the best rating. They also rated 3 possible features. For example, users gave a mean rating of 8.7 (standard deviation = 0.5) to the usefulness of pruning, a mean rating of 8.2 (standard deviation = 1.0) to the usefulness of dynamic queries, while the ease of range-slider manipulation got a mean rating of only 4.8 (standard deviation = 2.4).

5.4.3 Suggestions for improvement

Subjects were also asked to make suggestions for improvement and possible new features. All the suggestions were tabulated, resulting in 19 possible enhancements to the system. Some of these are obvious ones that would be inexpensive to incorporate into the system, while others would require more effort:

(1) Ability to manually add or remove 1 or more nodes to the selection subset resulting from the dynamic queries (by clicking, marquee etc.). Thus, the ideal subset specification mechanism would allow both query-based specification as well as manual specification. Some issues relating to keeping track of these nodes and avoiding complications later in the query process need to be addressed.

(2) Pruning and the Hide Gray Leaves feature were clearly well liked by the subjects. Most of them used the latter frequently to recapture screen space from uninteresting gray nodes and focus

the search. Removing nodes like this also made the system faster. Many subjects agreed that a Hide All Gray Nodes feature would be very useful. Some subjects also felt that the default behavior could be to hide gray nodes (when levels changed) and users could ask to see gray nodes (for context) if they were interested. This suggestion should help make the views even more compact. But the preferred default behavior depends on the structure of the tree (fan-out at each level) and also the type of tasks; therefore this requires more investigation. In fact, one subject liked seeing the gray nodes at higher levels and used the Hide Gray Leaves feature only once. He said the presence of the gray nodes at higher levels helped in visually separating disjoint groups of nodes.

(3) Subjects found the response time of some PDQ Tree-browser features to be slow, especially increasing levels when the number of nodes was high.

(4) It was observed that subjects get somewhat disoriented when the level of the tree was changed. This is because the layout algorithm generates a fresh layout whenever the tree structure changes, i.e. whenever more or less levels are requested to be seen. It is felt that this problem can be significantly alleviated by retaining the same current focus. For example, if the user asks to see the University level while the state Florida is near the center, the new view should be initialized to show universities within Florida.

(5) Panning the detailed view by dragging the field-of-view in the overview was found useful, but some improvements to the design are required. One subject emphasized the need to always fit the overview into one screen only, so that no scrolling of the overview is required. As mentioned before, this is what we had designed, but it wasn't enforced in this implementation. Another subject suggested that users should be able to click anywhere in the overview and have the field-of-view jump to that position. This would enable fast coarse navigation. Fine-tuning could then be accomplished by dragging the field-of-view.

(6) The catchiest quote received from one of the subjects was this: "Drag-and-drop becomes a drag for experienced users, so drop it!". Some other subjects also echoed the feeling that it might be easier and faster to just replace each drop area with a menu of attributes at that level. Some other subjects enjoyed the drag-and-drop mechanism to create and modify query panels.

(7) Ability to specify complex boolean queries involving ORs and NOTs, in addition to the ANDs allowed currently.

(8) One task (Task 6) required subjects to look for departments within 3 states, Wyoming, Wisconsin and Washington. This is basically restricting states to Wyoming OR Wisconsin OR Washington. The menu allows users to select 1-of-n textual values. A widget to select m-of-n textual values needs to be designed and implemented.

(9) One subject wished that there was a way to search upwards in the tree, i.e. to be able to query on universities and then see the states which contained the selected universities. Another subject made an interesting suggestion, that the PDQ Tree-browser should allow users to hide certain levels on demand. For example, if users are interested in looking at all universities in USA, and do not care about the states they are in, it should be possible to remove the State level totally and then get it back when desired.

6 A controlled experiment

With our guidance, three students, Robert Ross, Zhijun Zhang and Eun-Mi Choi, conducted a controlled experiment to compare 3 behavior alternatives for the PDQ Tree-browser (Ross, Zhang, & Choi, 1994). The UniversityFinder database was used in this experiment. These behavior alternatives related to pruning subtrees of nodes that did not match the query at their own levels. The 3 treatments were as follows: (two of these were discussed in Section 4.5)

Full-tree: Subtrees of unselected nodes are shown in their entirety, but are colored gray.
Partially-pruned: Subtrees of unselected nodes are pruned out but the nodes themselves are shown in gray (This is the behavior option that the PDQ Tree-browser currently uses).
Fully-pruned: Subtrees of unselected nodes and the unselected nodes themselves are pruned out.

It was hypothesized that subjects using the full-tree interface would take longer than those using the partially or fully-pruned interfaces. Although the full-tree allows for a static display of the tree, it's inclusion of all irrelevant nodes would probably slow down the response time, add too much useless information to the user's visual field, and create a need for excessive scrolling/panning of the displays to get to relevant nodes. Due to the longer task completion times and more difficult / complex searching required, users would also have lower subjective ratings for the full-tree interface than for the pruned interfaces. Further, the completion times and subjective ratings for the pruned interfaces would be approximately equal (differences would not be significant). It was postulated that the fully-pruned interface promised the most compact views, while the partially-pruned interface offered additional context feedback that might help in tasks requiring several iterations (to refine the query).

24 subjects were randomly assigned to use one (and only one) of these three treatments (between-groups experimental design). They performed a set of seven tasks. Afterwards, they filled out an electronic questionnaire (QUIS) to subjectively evaluate the treatment that they had used. The results (Tables 1 and 2) showed that the times to complete the set of seven tasks were significantly different at the 0.05 level with interface 1 being slower than 2, which was slower than 3. In terms of subjective satisfaction, 2 rated higher than 3, which was preferred to interface 1.

Full-Tree	**Partially-Pruned**	**Fully-Pruned**
1917 (609)	883 (121)	705 (139)

Table 1: Mean task completion times (in seconds) with standard deviations in parentheses

Full-Tree	**Partially-Pruned**	**Fully-Pruned**
0.42 (0.12)	0.71 (0.08)	0.64 (0.07)

Table 2: Mean subjective satisfaction ratings (normalized to 0 - 1, 1 being best) with standard deviations in parentheses

For total task completion time, an ANOVA gave $F(2, 21) = 25.2$ which was significant at the 0.01 level. Then, using pairwise t-tests, it was found that all results were significant at the 0.05 level. Therefore, in this controlled experiment, the fully-pruned interface was the fastest, while the full-tree interface was the slowest. The differences between the full-tree and the pruned interfaces were also significant at the 0.01 level, but the differences between the partially and fully-pruned interfaces were not. For subjective ratings also, the ANOVA produced an F that was significant at the 0.01 level. Also, all of the pairwise t-tests were significant at the 0.05 level. Users liked the partially-pruned interface the best and disliked the full-tree interface the most. Differences between the full-tree and the two pruned interfaces were also significant at the 0.01 level although differences between the partially and fully-pruned interfaces were not. In addition to the overall times for each interface, the times to complete each task were also compared.

The poor performance times of subjects using the full-tree interface might be partly due to poor system response times. Specifically, that interface had consistently (noticeably) slower response times, especially when expanding to the lower levels ("University" and "Department" levels).

The differences between the two pruned interfaces were not as clear cut; task performance was faster with the fully-pruned interface but the partially-pruned interface got higher subjective ratings. The fan-out of the particular tree used probably increased the differences between these two interfaces. Specifically, the UniversityFinder tree has a large fan-out (50 states in USA) from the Region level to the State level, and this might help explain why the fully-pruned interface had faster performance times than the partially-pruned interface (a significant proportion of states were gray and thus took up a significant proportion of screen space). It would be interesting to repeat the experiment with trees of varying fan-out.

This controlled experiment clearly showed the advantages of pruning. The choice of whether nodes not matching the query should be grayed or removed out is best provided via the interface to the users.

7 Conclusions

The PDQ Tree-browser visualization tool presents trees in two tightly-coupled views, one a detailed view and the other an overview. Users can use dynamic queries to filter nodes at each level of the tree. The dynamic query panels are user-customizable. Subtrees of unselected nodes are pruned out, a feature that usability testing and a controlled experiment, showed to be very useful. Possible enhancements to the PDQ Tree-browser were identified.

The concepts of dynamic querying and pruning are general enough that they can be applied effectively to other existing tree visualizations like treemaps and Cone Trees. These concepts are also extensible to graph structures, but that would require careful thinking and design.

7.1 Future Directions

(1) PDQ Tree-browser refinements:

- Implementing the improvements that were identified in the usability testing (Section 5.4.3).
- The PDQ Tree-browser interface has been ``fine-tuned'' for a tree of depth 5 (See Section 4.4 for details). The interface could be extended to cope with varying structure and growing size by following the suggestions in section 4.6.1 or otherwise.
- Further study of which approach is better in terms of hiding vs. showing gray nodes (at higher levels) by default is needed (to extend the work of Ross, *et al.*, 1994). This investigation should take into consideration different types of tasks, applications and tree structures (fan-out, depth, breadth, size) and attempt to identify when either approach will yield superior performance.

(2) General GUI extensions: Newer widgets need to be developed to enable users to specify multiple selections on textual attributes easily. Another useful widget would be an extension of the 2-box slider to a n-box slider.
(3) Extension to Graph Structures: The PDQ Tree-browser extended dynamic queries to one class of non-flat data sets, i.e. hierarchical data sets. The most general form of any data set, is the arbitrary graph. Graphs have nodes connected via an arbitrary number of links of different types. Each link represents a relationship between (among) the connected nodes. We believe that PDQ Tree-browser concepts of dynamic querying and pruning / selective growing can be extended to graph structures as well. Instead of distinct levels in the hierarchy, we would then think of distinct classes of nodes (and even links).

7.2 Other tree-browsing research issues

(1) Semantics-based browsing: Generic 2D browsers (Plaisant, *et al.*, 1995) treat the information space being browsed as images only. We believe that browsing of trees can be facilitated by taking advantage of the underlying structure of the tree. Fast navigation between siblings, up to parents and grand-parents etc., without having to manually scroll and pan would be useful options. Traversal of the tree in preorder, postorder and inorder, and tours of nodes marked either manually or by a query are interesting topics for investigation.
(2) Layout issues: There is a clear need for layout guidelines for tightly-coupled overviews and detailed views, as the size and fan-out of trees vary. For example, if the tree is wider than deep (as was the case in the UniversityFinder scenario) then it makes sense to have the tree drawn from left-to-right (or right-to-left) and the overview to the left (or right) of the detailed view. On the other hand, if the tree is deeper than it is wide, then it might be better to have the tree drawn from top to bottom and the overview below the detailed view. If the tree structure changes frequently and has to be redrawn, then it might be a good strategy to utilize the overview optimally by making the remaining tree occupy the entire overview space. But this might lead to some disorientation, as the zoom ratio will keep varying.

Acknowledgments

We are thankful to Dr. Michael Ball for his insightful suggestions, to Robert Ross, Zhijun Zhang and Eun-Mi Choi for designing and running the controlled experiment that was described in this paper, and to all the usability testing subjects who provided us with invaluable feedback. Thanks also to Edward Johnson for suggesting the name "PDQ Trees" for our system.

This project was supported in part by Hughes Network Systems, Maryland Industrial Partnerships (MIPS), the Center for Satellite and Hybrid Communication Networks, and the National Science Foundation grants NSFD CD 8803012 and NSF-EEC 94-02384.

References

AHLBERG, C., SHNEIDERMAN, B. (1994). Visual Information Seeking: Tight Coupling of Dynamic Queries with Starfield Displays. *Proceedings of the ACM Conference on Human Factors in Computing Systems,* pp. 313-317, ACM., New York.

BATTISTA, G. D., EADES, P., TAMASSIA, R., TOLLIS, I.G. (1989). *Algorithms for drawing graphs: An annotated bibliography*. Unpublished Technical Report, Brown University, Computer Science Department, October.

BEARD, D. V. , WALKER II, J. Q.(1990). Navigational Techniques to Improve the Display of Large Two-Dimensional Spaces. *Behavior & Information Technology*. **9** (6), 451-466.

BUJA, A., MCDONALD, J. A., MICHALAK, J., STUETZLE, W. (1991). Interactive Data Visualization using Focusing and Linking. *Proceedings of the IEEE Visualization '91 Conference*, pp. 156-163, San Diego, CA, IEEE Computer Society Press, October .

CARR, D., JOG, N. K., KUMAR, H., TEITTINEN, M., AHLBERG, C. (1994). Using interaction object graphs to specify and develop graphical widgets. Technical report CS-TR-3344, University of Maryland, Department of Computer Science, September.

CHIGNELL, M. H. , ZUBEREC, S., POBLETE, F. (1993). An Exploration in the Design Space of Three Dimensional Hierarchies. *Proceedings of the Human Factors Society,*, Santa Monica, CA.

GEDYE, D. (1988). *Browsing the tangled web*. Master's thesis report, University of California at Berkeley, Division of Computer Science, May.

HENRY, T. (1992). *Interactive Graph Layout: the Exploration of Large Graphs.* Ph.D. Thesis,, University of Arizona, Tucson, AZ, May.

HENRY, T. R., HUDSON, S. E. (1991). Interactive Graph Layout. *In Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pp. 55-64, ACM, New York, November.

HOLLANDS, J. G., CAREY, T. T., MATTHEWS, M. L., MCCANN, C. A. (1989). Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views. In G. Salvendy and M.J.Smith, Eds., *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, pp. 313-320, Amsterdam:: Elsevier Science.

KUMAR, H. (1995). Visualizing Hierarchical Data with Dynamic Queries and Pruning - The Tree-browser. In Catherine Plaisant, Ed., *HCIL Open House '95 Video*, Human-Computer Interaction Laboratory, University of Maryland, June.

KUMAR, H., PLAISANT, C., TEITTINEN, M., SHNEIDERMAN, B. (1994) Visual *Information Management for Network Configuration.* Technical Report CS-TR-3288, University of Maryland, Department of Computer Science, June.

MARCHIONINI, G. (1995). *Information Seeking in Electronic Environments*, Cambridge University Press, UK.

PLAISANT, C., CARR, D., HASEGAWA, H. (1992). *When an intermediate view matters - A 2D-browser experiment.* Technical Report CS-TR-2980, University of Maryland Department of Computer Science, October.

PLAISANT, C., CARR, D., SHNEIDERMAN, B. (1995). Image Browser Taxonomy and Guidelines for Designers, *IEEE Software* **12**, (2), 21-32, March.

ROBERTSON, G. G., MACKINLAY, J. D., CARD, S. K. (1991). Cone Trees: Animated 3D Visualizations of Hierarchical Information. *Proceedings of the ACM Conference on Human Factors in Computing Systems,* pp. 189-194, New York:ACM.

ROSS, R., ZHANG, Z., CHOI, E. (1994). *Using the Tree-browser to visualize information for database queries.* Unpublished report, University of Maryland, Department of Computer Science, December.

SCHAFFER, D., ZUO, Z., BARTRUM, L., DILL, J., DUBS, S., GREENBERG, S., ROSEMAN, M. (1996). Comparing Fisheye and Full-Zoom Techniques for Navigation of Hierarchically Clustered Networks. *ACM Transactions on Information Systems* **14**, (2).

SHNEIDERMAN, B. (1992). Tree Visualization with Treemaps: 2-d Space-Filling Approach., *ACM Transactions on Graphics*, **11**, (1), pp. 92-99, January.

SHNEIDERMAN, B. (1994). Dynamic queries for visual information seeking. *IEEE Software*, **11,** (6), 70-77.

YOUNG, D., SHNEIDERMAN, B. (1993). A graphical filter/flow representation of boolean queries: A prototype implementation and evaluation. *Journal of the American Society for Information Science*, **44**, (6), pp. 327-339, February.

Biographies

Harsha Prem Kumar received a B.Tech. in Textile Engineering from the Indian Institute of Technology, New Delhi, India in 1992, and an M.S. in Systems Engineering from the University of Maryland at College Park, in 1994. He is currently a member of the Usability Engineering group at Bellcore, Inc., Piscataway, New Jersey. His research interests include information visualization, user interfaces for network management, and interactive information systems.

Catherine Plaisant is assistant research scientist at the Human-Computer Interaction Laboratory of the Center for Automation Research at the University of Maryland. Her work centers primarily on graphical user interface design and evaluation. She has worked on various research projects related to hypertext, touchscreens, public access information systems, information visualization and many other domains. Plaisant received a general engineering degree from l'Ecole Nationale Supérieure des Arts et Métiers in Paris France, and a Docteur Ingénieur degree from the Université Pierre et Marie Curie in Paris.

Ben Shneiderman is a professor of computer science, head of the Human-Computer Interaction Laboratory and a member of the Institute of Systems Research, all at the University of Maryland at College Park. He organizes an annual satellite television presentation, "User Interface Strategies," is the author of *Software Psychology: Human Factors in Computer and Information Systems* (Little Brown, 1980) and *Designing the User Interface: Strategies for Effective Human-Computer Interaction: Third Edition* (Addison-Wesley, 1997) and the editor of *Sparks of Innovation in Human-Computer Interaction* (Ablex, 1993). Shneiderman received his Ph.D in computer Science from the University of New York at Stony Brook.

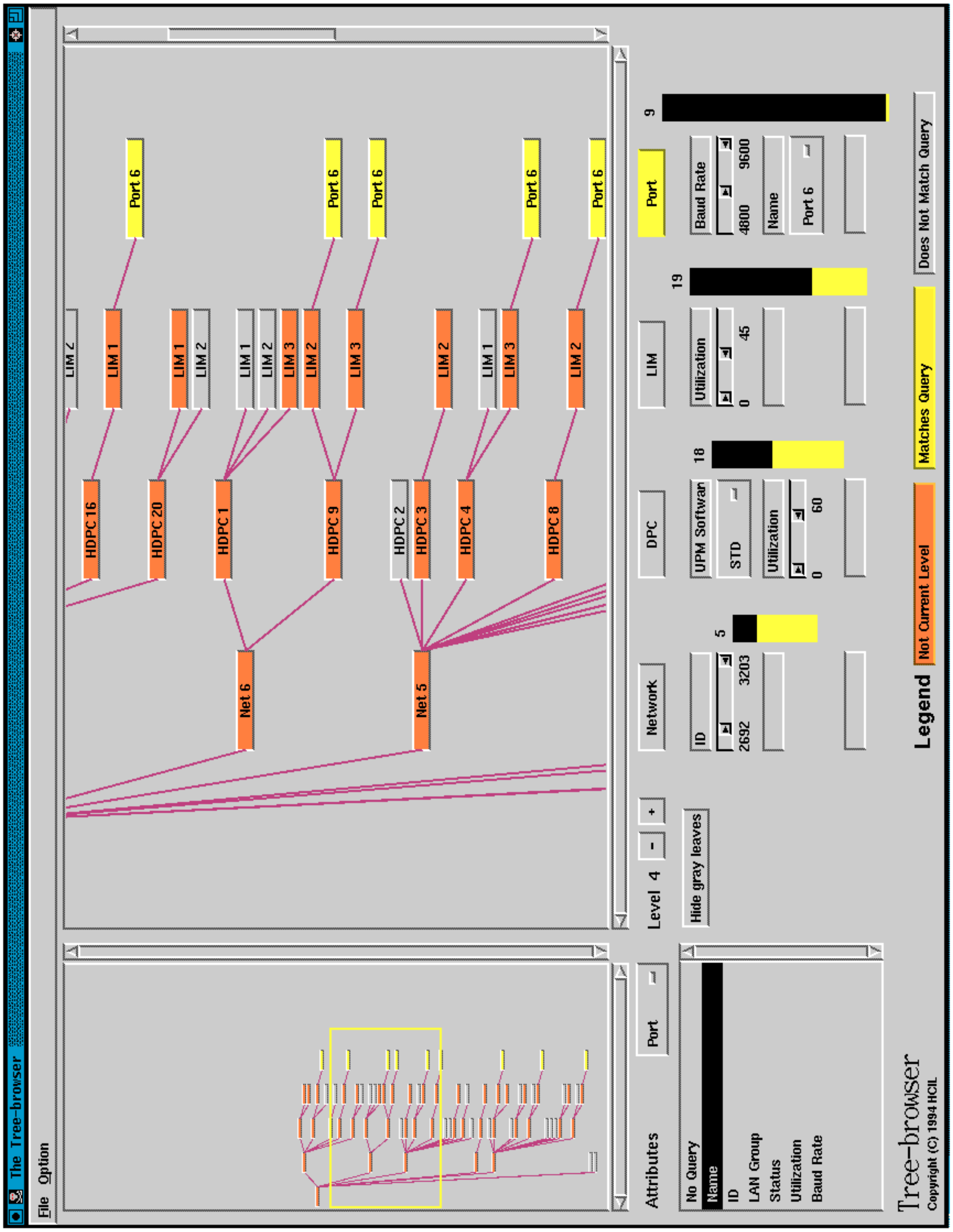


Figure 1: the PDQ Tree-browser interface

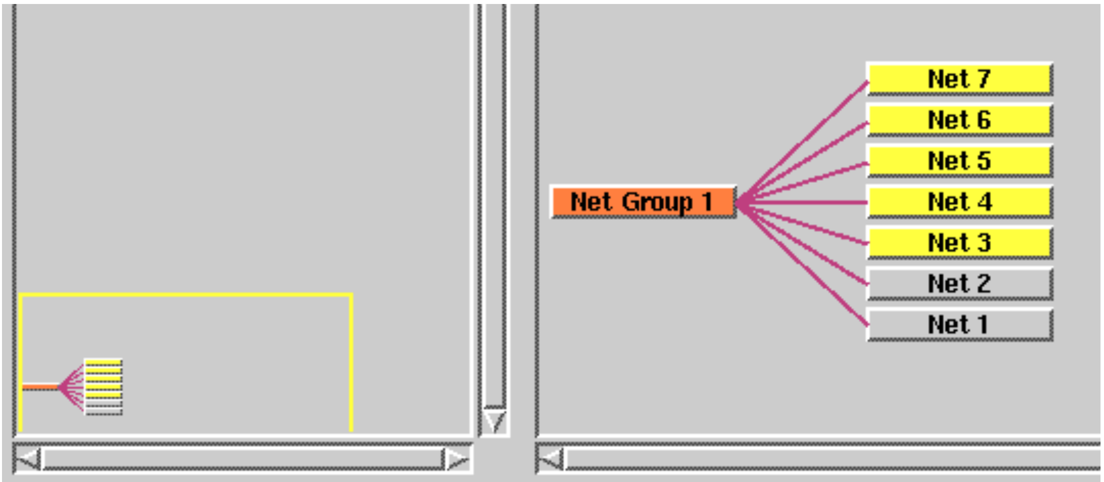


Figure 2: Nets 1 and 2 do not match query

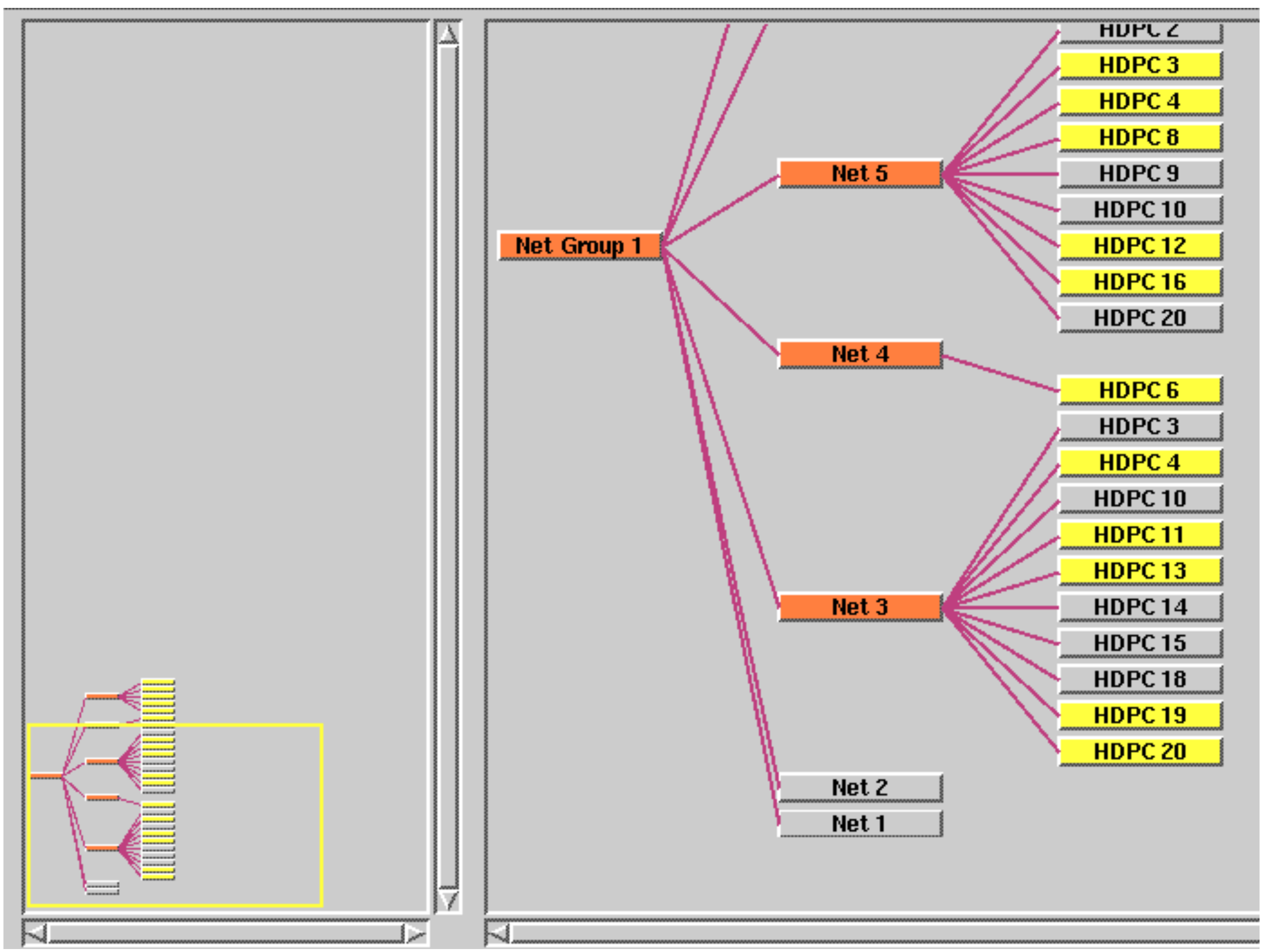


Figure 3: Subtrees of Nets 1 and 2 are pruned out

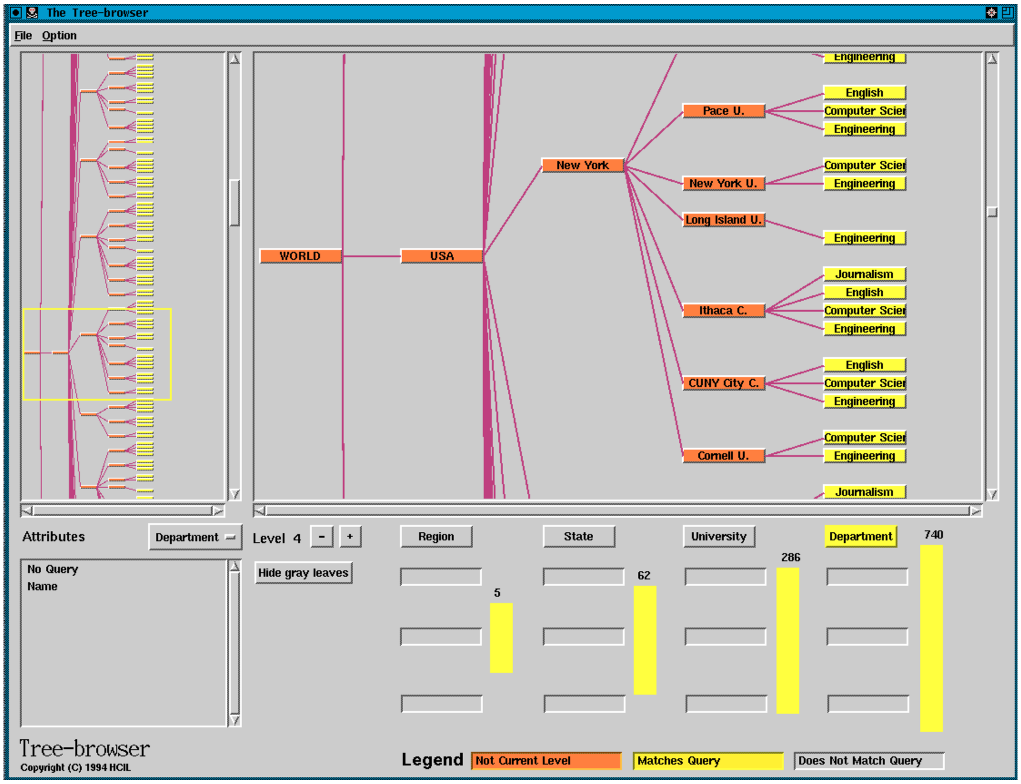


Figure 4: PDQ Tree-browser applied to the University Finder application

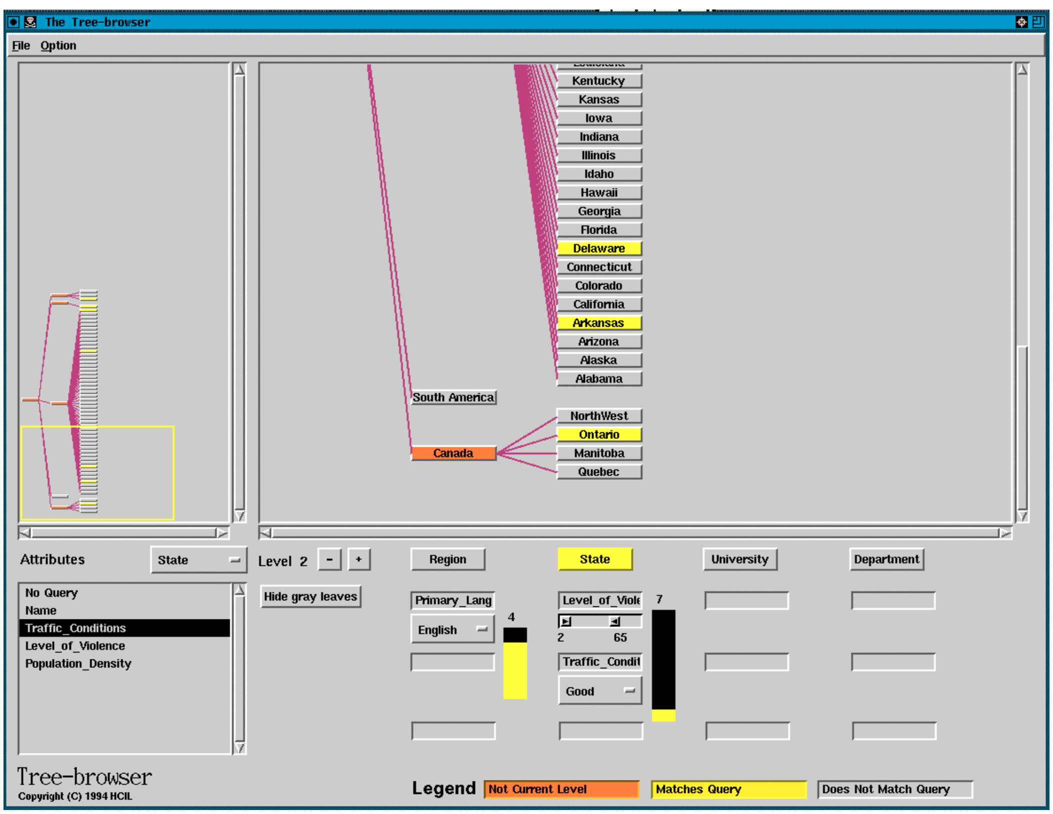


Figure 5: Querying for desirable states

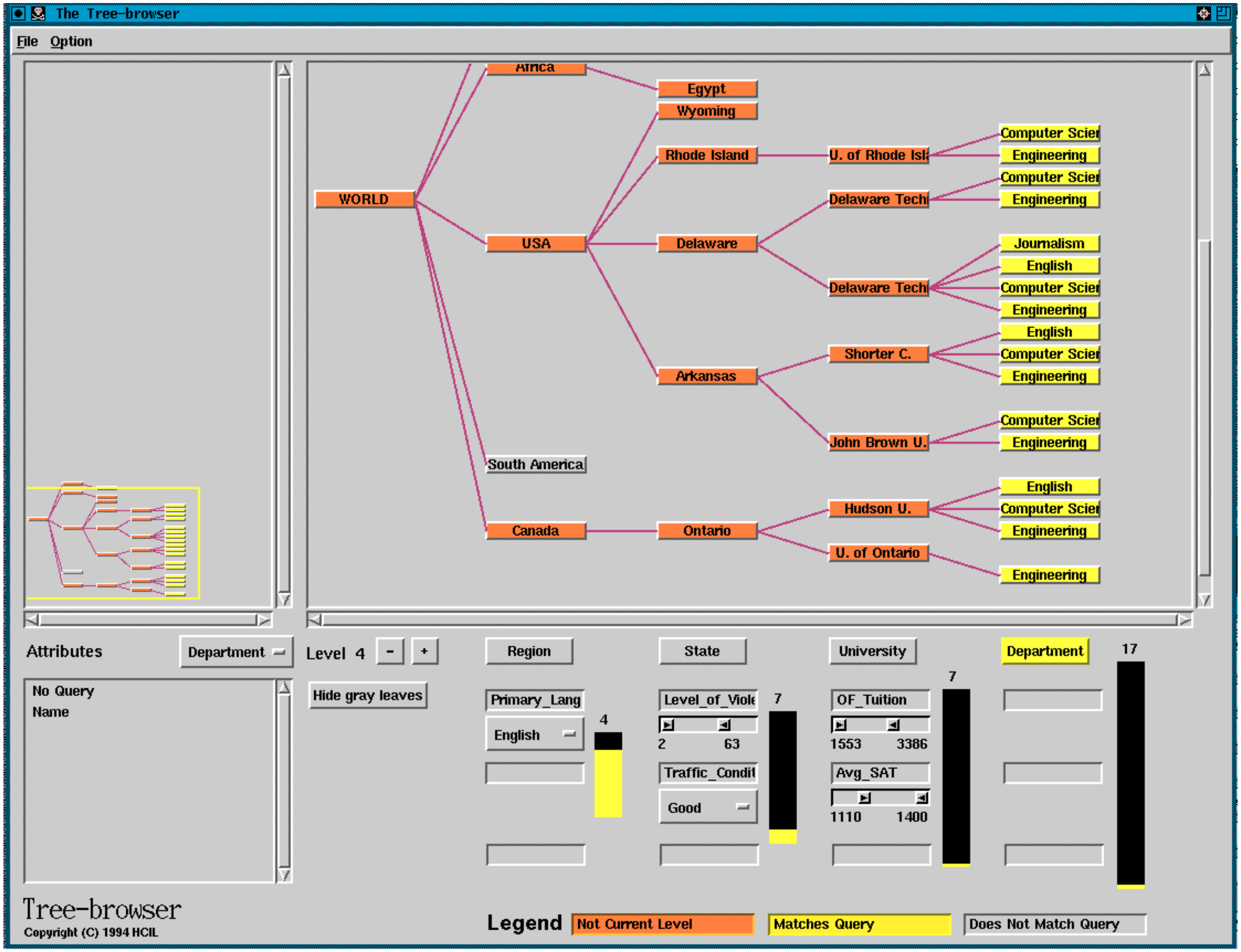


Figure 6: Final selection sub set of 17 universities

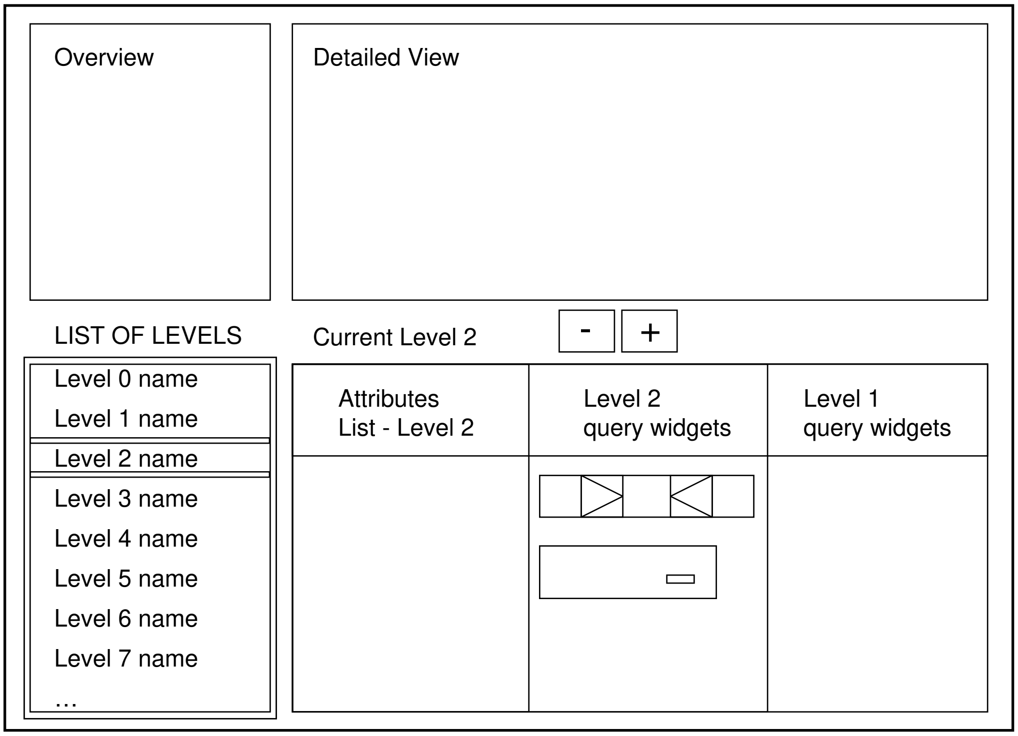
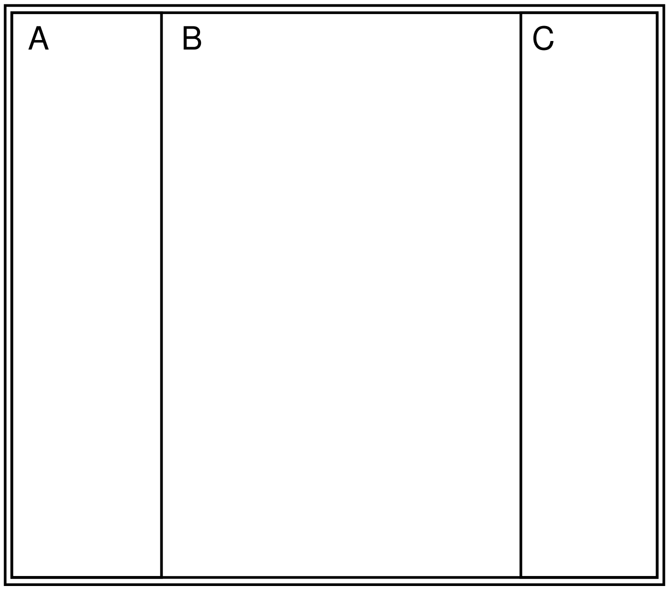
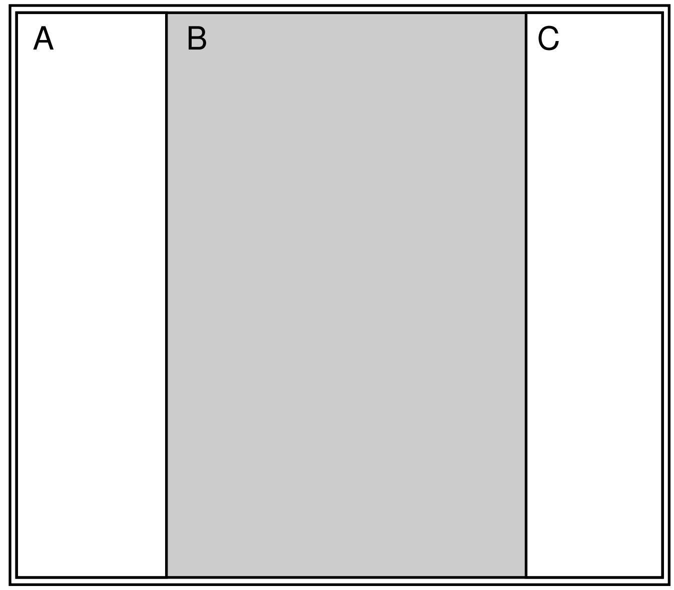


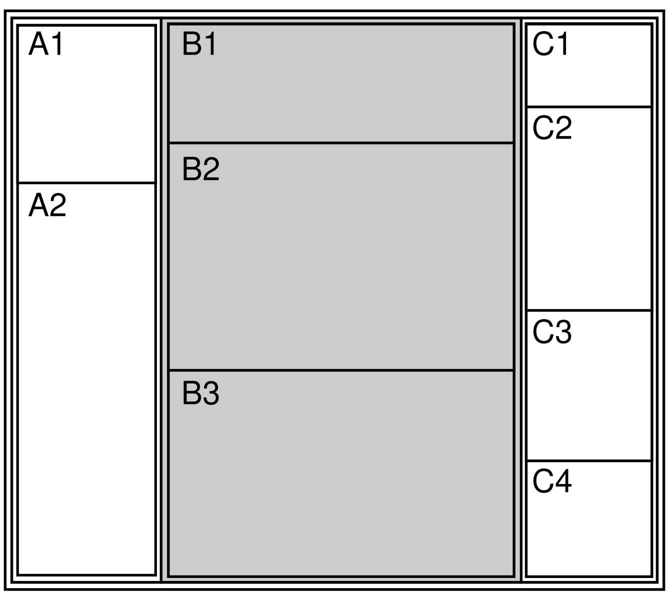
Figure 7: Extending the PDQ Tree-browser design to trees of arbitrary depth



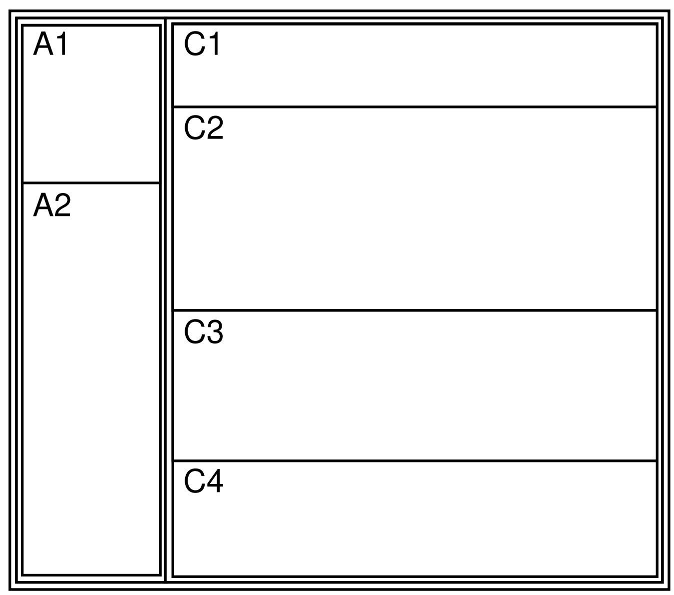
(a) Level 1



(b) Level 1 dynamically queried



(c) Level 2 without pruning



(d) Level 2 with pruning

Figure 8: Pruning applied to treemaps