Multitrees: Enriching and Reusing Hierarchical Structure

George W. Furnas and Jeff Zacks

Computer Science Research
Bell Communications Research
445 South Street, Morristown NJ, 07960
Tel: +1 (201) 829-4289 Tel: +1 (201) 829-4320
E-mail: gwf@bellcore.com *E-mail: zacks@bellcore.com*

ABSTRACT

This paper introduces multitrees, a new type of structure for representing information. Multitrees are a class of directed acyclic graphs (DAGs) with the unusual property that they have large easily identifiable substructures that are trees. These subtrees have a natural semantic interpretation providing alternate hierarchical contexts for information, as well as providing a natural model for hierarchical reuse. The numerous trees found within multitrees also afford familiar, tree-based graphical interactions.

KEYWORDS: Information graphs, representation, hierarchies, reuse, directed graphs, hypertext structures, graphical browsers

INTRODUCTION - REPRESENTATION AND REUSE

Two motivations came together in this work. The first was a long standing interest in finding structures for representing information which might be richer than trees yet still viewable and navigable. Trees have a long history in representing information, from the Dewey Decimal system and tables of contents of books, to their use in online information systems and electronic document delivery systems such as Superbook^{tm}[3]. Trees have many strengths. They can be laid out nicely in the plane, they allow simple complete traversal algorithms, they offer a natural analog for the semantic notions of abstraction and aggregation. Trees also have weaknesses. They allow only one way to get from one node to another so there can be no shortcuts, no alternative organizations. In contrast, the fully general graphs often used in hypertext systems have complementary strengths and weaknesses. They allow many routes between things: cross references, multiple organizing contexts, etc., but the structures are not easily laid out, users are easily lost and abstraction is not well represented. Somewhere in between are directed acyclic graphs (DAGs), whose directed links are constrained to have no cycles. DAGs were proposed a decade ago[7] as an alternative information access mechanism. Like trees, the more general DAGs can represent semantic notions of abstraction: classes above subclasses, aggregations above subparts. The network of ISA relations of a knowledge base ("a dog ISA carnivore") and the rich multiple inheritance structures of object oriented systems form DAGs that capitalize on this semantics. For information access, these structures support a top down search strategy like trees, a natural orientation and abstraction mechanism. A problem with DAGs, however, is that like general graphs, they are perhaps underconstrained. In particular, they can be quite difficult to lay out and comprehend. Even small neighborhoods can easily be non-planar causing many edge crossings in viewers. (See [8] for a discussion of some of these topological issues and their implications for interfaces.)

Our first motivation, then, was the question: are there any other places of interest along this spectrum of structure from trees to graphs? In particular, there might be a class of structures in between trees and DAGs, that could at least offer new options in the design space for information representation.

Our second, very concrete motivation came from issues of reuse of hierarchical structure. Under a company-wide initiative huge collections of documentation were being assembled into a large online information resource. The whole assembly formed a great tree of information, with some hierarchy imposed to organize the collection of documents at the higher levels, and lower levels of structure coming from within the documents themselves. This structuring reflected a tentative acceptance of the advantages of the tree approximation of the universe, with its comparative comprehensibility, viewability, and navigability. Individual users, however, were having two sorts of difficulties. For one, the fixed structure of the tree organizing the documentation was indeed not suitable for everyone. As a result some alternative hierarchical organizations of the documentation were made available. Secondly users had trouble with the scale of this aggregate structure. They wanted smaller views, to allow quick access to those parts of the large tree that they needed for their tasks. Some devices were created to allow users to personalize their views of the database. They could collect documents on a virtual "bookshelf" and compile sets of pointers to document sections into lists of "bookmarks." Note however that both of these

CHI94-4/94 Boston, Massachusetts USA

techniques collect fragments of the existing structure only into unstructured lists. One might easily imagine users wanting to organize these sets of fragments into their own private trees, to suit their own purposes. Thus these two problems and their solutions have in common the notion of allowing fragments of some existing tree to be reused, reorganized into new trees. This simple idea turns out to have intriguing consequences.

This paper explores those consequences by introducing a new class of structures called *multitrees*. We will first illustrate the idea of a multitree with a concrete construction scenario. We will then explore some of the formal properties of such structures, particularly properties relevant to interfaces, and illustrate them with sample views from prototype browsers we have built. Then we will question and discuss the constraints imposed by multitrees, finishing with a general discussion.

AN INITIAL EXAMPLE: MULTITREES AND THE COLLEGE PROFESSOR

Reuse of hierarchical structure is not new. It occurs, for example, whenever a college professor creates a structured syllabus of readings for her course. Instead of writing a new document from scratch, she selects fragments from the existing body of literature. For the purposes here we let that body of literature be represented by a great tree, with organization at higher levels coming from, for example, the Dewey Decimal system or the Library of Congress classification system, and the lower levels from the internal hierarchical structure of chapters and sections within documents. From this large tree (Figure 1(i)), Professor A pieces together her own structure of course readings: a book from here, a chapter from there, a section from somewhere else, perhaps even a few short volumes from somewhere. Consider her new structure along with the original tree (Figure 1(ii)). What she has done is to select nodes from the original structure, together with their dependent subtrees, and spin a new hierarchical superstructure, a new tree, above those pieces. A second professor, B, could build another such structure, perhaps even using whole pieces from A's syllabus, or writing and including entirely new tree fragments. The only constraint is that each syllabus be a tree, i.e., it must start from disjoint tree fragments, and assemble them together into some new hierarchy.

The resulting overall structure we call a *multitree*. It has the unusual property that although it is not a tree, the descendents of any node form a tree. Thus familiar tree presentation and navigation techniques may be used to view large fragments of the structure. Perhaps one of the simplest examples of such a structure is when one has two trees sharing the same set of leaves, for example the Dewey Decimal and Library of Congress classification systems for books. General multitrees, however, may share complete subtrees, not just leaves. (By complete subtree we mean a node and all its descendents). That is, hierarchical structure can be shared at multiple granularities.

In hierarchical reuse, more can be shared than just the text itself. Joel Remde (personal communication) has been exploring the use of hierarchical indices for huge simple hierarchies of text. Each node in the simple tree structure has an associated index indicating for each word which immediate descendent node represents a text segment containing that word. One follows a chain of such pointers from index to subindex to find actual occurrences in the leaves. There are various advantages for updating, distributing the database, limiting the scope of searches, and displaying high level results against views of the hierarchy. In multitrees, such hierarchical indices can be shared as well. When Professor A includes a chapter in her syllabus, she gains access to the index for that chapter as well.

Multi-trees are DAGs, not trees, and as a result a node in the structure can have multiple parents. In fact it can therefore have multiple ancestral lineages: one for each tree that (re)used it. Thus looking upward from a node one can see the diverse hierarchical contexts in which the node has appeared. So, whereas looking downward one sees the various ***contents*** under a node (subsection 1, subsection 2, etc.), looking upward one sees its alternative ***contexts*** (as a chapter in Tree1, as Part 1b of Professor A's course, etc.)

There are several further uses for these multiple contexts. For example they can support a special kind of browsing. Suppose Professor A teaches a course on algorithms and includes a particular paper in her syllabus. Using the global

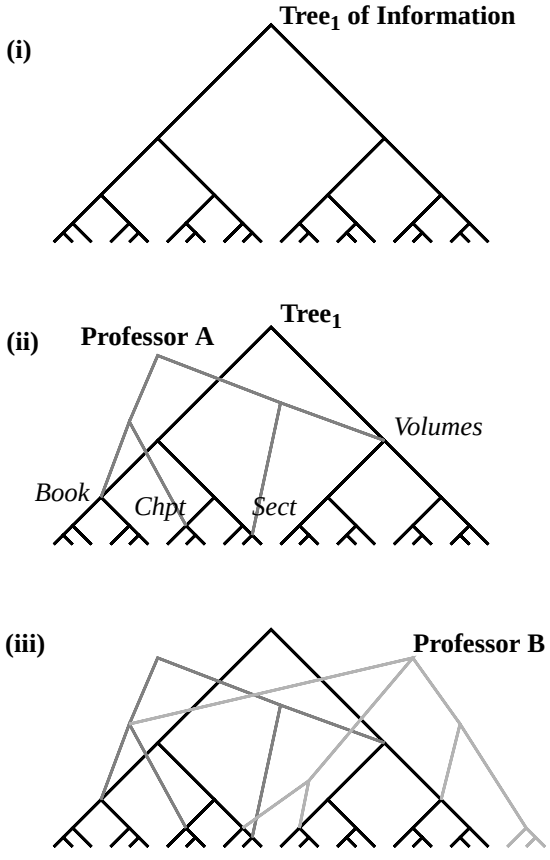

Figure 1 . Constructing a multitree - an example. Beginning with an initial tree of information, Professor A builds her course syllabus. In (a) she chooses a disjoint set of complete subtrees and adds new tree structure above them. Similarly, Professor B makes another, chooses fragments, including some of A's and some completely new ones, and builds a tree above to organize them.

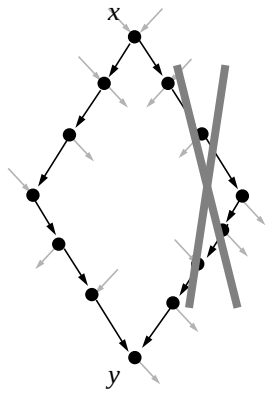

Figure 2 . Diamonds are not permitted in a multitree. A diamond occurs when two distinct directed paths occur between a pair of nodes. Thus in a multitree at most one directed path can exist between two nodes.

structure, she may look upward from that paper to find some other syllabus which also used that paper, move to the context of that syllabus and browse within it to get ideas of other papers to assign.

Contexts can also be used for conditional inheritance. One could, for example, make the presentation style of a node be conditional on the selection of a particular context. Thus in Figure 1 one would select for viewing not just some particular chapter sized unit of text, but also indicate the context in which it is being viewed (e.g., Professor A's course). The formatted look and feel of the segment of text would be in part inherited from the selected context and altered accordingly. Thus material is reused but can be modified by the structural context.

These comments on multiple parents and their interpretation as contexts would apply to general DAGs as well. One difference is that the contexts (i.e., ancestor nodes) all have simple hierarchies beneath them. Thus looking upwards one sees in effect a choice, not of arbitrary ancestral nodes, but of possible trees to be browsed. There is in fact special structure to this set of choices, arising from a structural duality in multitrees, to which we turn now.

DIAMONDS, DUALITY AND TREES ALL OVER

There are several ways to define the class of multitrees, two of which have been given so far. One was procedural, based on building new trees above disjoint tree fragments. A resulting declarative characterization was that multitrees are DAGs whose nodes have descendents forming only trees. A new formulation equivalent to this last is to say that these DAGs have no downward diamonds. A downward diamond is formed when two downward paths diverge from some node, *x*, and meet again some node, *y*, below (see Figure 2). Clearly if such a diamond exists, then the descendents of *x* do not form a tree, and if no such diamond exists they do. This formulation is interesting because it is what is called self-dual, that is it also holds for the links in the DAG considered in the other direction: there are no downward diamonds *iff* there are no upward diamonds. This in turn has the rather surprising consequence that not only does the set of *descendents* of any node form a tree, the set of its *ancestors* also form a tree (an inverted tree). Several such trees can be seen in Figure 1(iii), starting at a leaf and following paths upward that bifurcate to reach different roots. In summary,

Proposition 1. *The following properties are equivalent:*

(a) The DAG can be constructed by adding new tree structure above existing (or newly added) disjoint complete subtrees.

(b) The descendents of any node form a tree.

(c) The DAG is diamond free.

(d) The ancestors of any node form an inverted tree

Any DAG satisfying these conditions is called a multitree.

In this light it is interesting to note that genealogies, though often called "family trees", are in fact not trees at all. Except for intermarriage, which would introduce diamonds, they are multitrees. The property just derived makes sense of the fact that a given individual has both a tree of ancestors and a tree of descendants within the structure. In fact we have found it a useful exercise to represent our own family genealogies with these multitrees, and to test our browsers on them.

For information representation, Proposition 1 means that looking down from any node in the structure one sees a tree of contents, and looking up one sees a tree of contexts. As mentioned already, each of the ancestral contexts itself roots a downward tree. Thus, although the structure itself is not a tree, and not planar in general, it has remarkably large easily identifiable and semantically coherent subsets that are trees, and hence are planar, and amenable to easy layout.

In fact the ubiquity of trees in this structure goes even further. So far, the word "tree" has been used in the common computer science sense of rooted tree (with directed links always pointing away from the root). Following the common graph-theoretic sense of tree, we will say that an undirected graph is what we shall call here a "topological tree" or "t-tree" *iff* it is a connected graph without cycles. T-trees are important since they too are always planar, and allow nice layouts and familiar browsers. Note that though multitrees have no diamonds (a condition on their directed links), when considered undirected they do have cycles (e.g., in Figure 1(ii), the cycle from *Book* to *ProfessorA* to *Volumes* to the $Tree_1$ root back to *Book*), and so they are not topological trees. Yet multitrees do have large topological trees within them, as described in the following proposition. Let us say $x \geq y$ iff x is above y in the DAG.

Proposition 2*. Consider any two nodes $x \geq y$ in a multitree, and the necessarily unique path connecting them. The union of all the ancestors of this path and all the descendents of this path is a topological tree.*

This result is illustrated in Figure 3 and has several practical applications for making useful graphical interfaces for multitrees. We consider some of these in the next section, with illustrations from screendumps of our prototype browsers written in Tk/Tcl.

VIEWING AND BROWSING MULTITREES

First consider when $x=y$. Then we have the simple case of showing simultaneously the tree of descendent contents and the tree of ancestral contexts for an individual node. We call

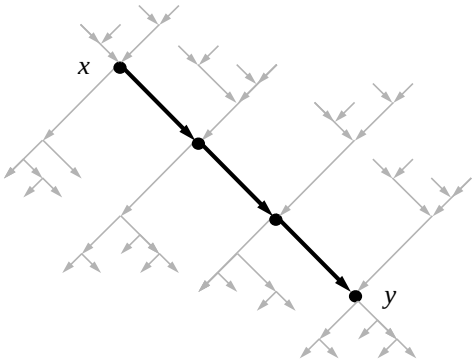

Figure 3 . Proposition 2 illustrated. Two nodes in a multitree, the unique path connecting them, and all the descendents and ancestors of this path together are guaranteed to form a topological tree.

this a *centrifugal* view of the structure, because it follows links in the direction away from some central point of focus.

Figure 4 shows our browser's centrifugal view of a dataset of local Bellcore information. This data set was drawn from a database maintained on our local file system at Bellcore. Users deposit useful information in a particular directory on an ad hoc basis. Anyone with an account is free to add or reorganize material. People have posted material related to corporate practices, life in Northern New Jersey, practical information such as directions, and diatribes on art. To create the multitree shown in these figures, the authors and one volunteer created new hierarchical organizations (trees) that used varying amounts of the pre-existing structure (the "original" tree from the file system). In Figure 4, the central highlighted cell is labeled "Directions." Looking down from the "Directions" node (to the right in this layout) we see the tree of files relating to how one travels to and from various Bellcore locations and related places. Looking up from "Directions" (to the left) we see that "Directions" is a descendent of the original tree and of the three post hoc classifications. Proposition 2 guarantees that both the upward and downward views are trees and that they can be shown simultaneously as single large t-tree. (Note that a centrifugal view may be truncated at any desired radius.)

An interesting consequence of Proposition 2 is that transitions between centrifugal views can be animated nicely. Consider browsing over multitrees using a centrifugal viewer and moving the focus around one link at time, say from node *a* to neighboring node *b*. In such a case, if *a* and *b* are adjacent, then one is above the other, and Proposition 2 applies, meaning that the union of the old and new views is in fact a t-tree, and can be displayed together in 2-D. This allows a nice animation option, showing the transition by first adding the new material, showing both together, then deleting the old.[1]

A third consequence of proposition 2 is for presenting various kinds of fisheye views of multitrees. Furnas[5][6] intro-

1. In practice there is some subtlety in that one might also want layouts to respect sibling order and direction of links (e.g., parents to the left of children).

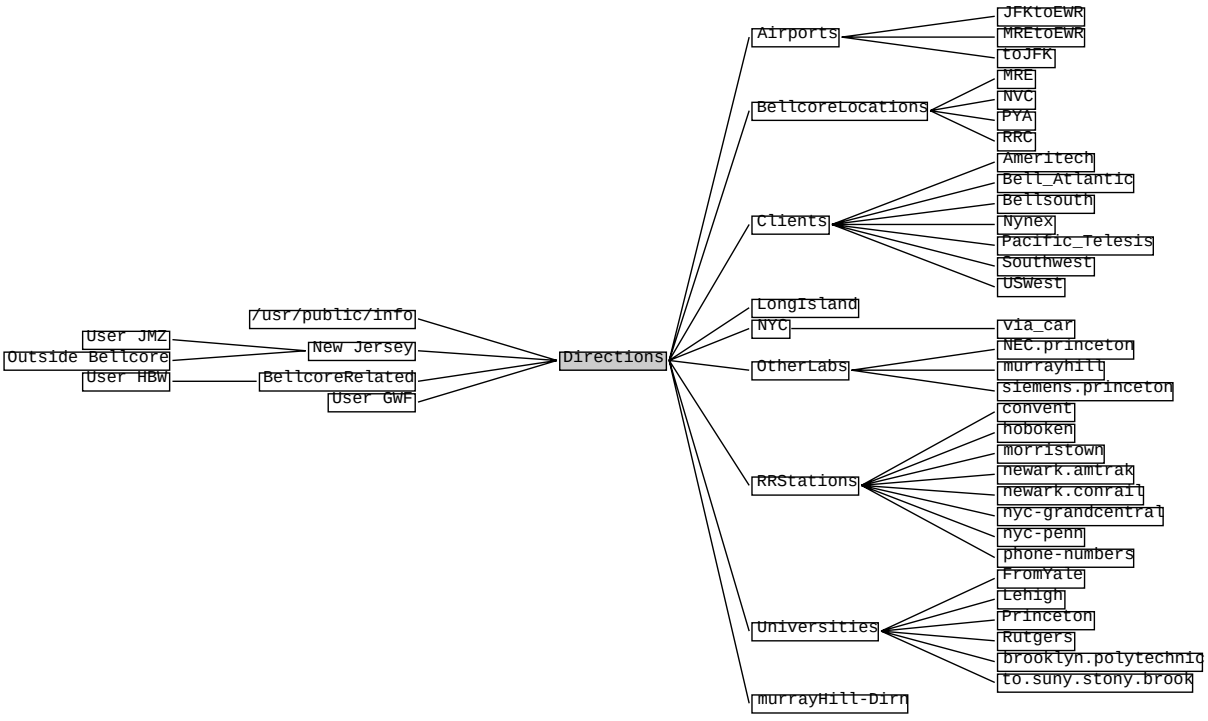

Figure 4 . Centrifugal view of the multitree built upon our /usr/public/info information repository. The view is centered on the node called "Directions" and shows the tree of ancestors (to the left) and the tree of descendents (to the right) of this node.

duced a notion of generalized fisheye views that tried to obtain a balance of local detail and global context. For simple trees it generated a scheme that assigned greatest interest to the ancestral lineage of a focal node, and next interest to nodes one link off that lineage, etc. A similar strategy is possible for multitrees and only requires that one specify not just a focal content node but a focal context node as well.Thus, for example, one could specify not just a leaf (as the focal content), but also a root (as focal context). The root selects a tree, and then an ordinary tree fisheye view can be generated. In the world of multitrees, however, there are a few new twists. For example one can shift around not only the leaf focus, but also the root focus, slowly changing contexts. A second interesting twist arises from the fact that just as there is a tree of contents from any chosen root, there is also an inverted tree of contexts looking upward from any leaf. Thus this inverted tree can also be viewed with a fisheye viewer -- so two fisheye views are possible for any chosen focal-context/focal-content pair.

Figure 5 shows a coordinated pair of first-order fisheye views for the same Bellcore information structure shown in Figure 4. The context focus is on one of the post-hoc organizations of the database and the content focus on the "Directions," both described above. The nodes in the lineage connecting them appear in black, the rest in gray. In the left pane we see the fisheye view of descendent trees; in the right view, the fisheye view of ancestor trees. There are pure indent-based layouts of both of these trees. Ordinary indenting used in familiar outlines is possible for the tree of descendents (a typical fisheye outline view). A complementary "outdenting" structure can be used for the tree of ancestors. This latter view takes getting used to so we have explicitly also drawn in the tree links to these views to make the structure more apparent.

Thanks to Proposition 2 it is possible to make a single view which shows both fisheye views at once: the lineage connecting a selected leaf and selected root, together with immediate ancestors and descendents of that lineage. Pretty versions of this, however, require one to be able to permute the ordering of branches, as is the case for the genealogy shown in Figure 6.

PROBLEMS AND SOME SOLUTIONS

Like other points on the spectrum of structural complexity from trees to graphs, multitrees have their own limitations. In this section we discuss some of these and related problems, as well as some strategies for working around them.

If diamonds are forever...

Multitrees are essentially defined by the absence of diamonds. This is a strong requirement and deserves more discussion. The existence of a single downward path from a node to a descendent is a familiar constraint on simple trees. It is a constraint we have often learned to live with, reflecting the ideal that the structure is at each level a partition. The tree successively refines the disjoint categorization of the world. If this ideal has merit then multitree structures offer a useful extension -- merely providing a family of alternative classifications in a single structure. (In fact, as it turned out from Proposition 1(c), tree-structured families of such classifications.)

Diamonds, however will not always go away. There are several circumstances under which we might want to consider

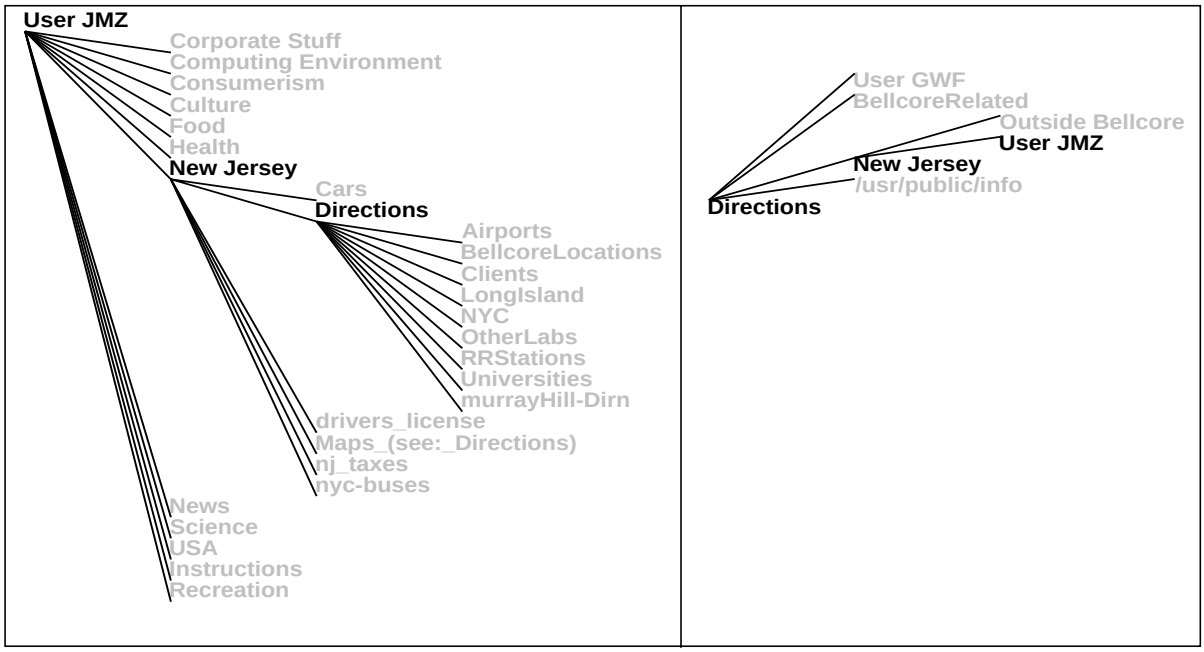

Figure 5 . Coordinated pair of first order fisheye views. Both views are defined by a focal-context, focal-content pair, here "User JMZ" and "Directions". The view on the left is a simple tree fisheye view of the downward tree of contents rooted at the context "User JMZ". It shows the ancestral lineage linking the focal-context and the focal-content, and the first-order descendents of the lineage. The view on the right is a similar tree fisheye view of the *inverted* tree of *contexts* rooted at the content "Directions". It too shows the ancestral lineage linking the focal-context and the focal-content, but this time with the first-order ancestors (contexts) of the lineage.

relaxing the diamond-free constraint. One case comes directly from the simpler world of trees, where from time to time, people want to put the same item in more than one place in the tree. Such violations of tree structure introduce diamonds and would similarly violate multitree structure. If diamonds are infrequent, we believe, this can be handled as with trees: The fundamental tree or multitree ideal is preserved, and some small exceptions are made, e.g., by virtual duplication of nodes. Furthermore if diamonds are long, as is often the case, then the structures are still locally multitrees, and local views will still behave ideally.

A more systematic problem arises when one has an internal node below which one would like to offer a choice of organization for descendents. For example one might have a subset of documents to be organized for access both alphabetically and by date. A whole suite of diamonds would begin at this node, and end at each of the cross-classified documents. For such cases, one might consider a new class of structures slightly more general than multitrees but from which sub-multitrees can be easily extracted. One would just need a way to break the diamonds in a systematic way to generate tree views as needed. For example, suppose any such node would be flagged (e.g., colored red) to indicate that its children offer two alternative breakdowns of its descendents, and that one must be chosen to preserve tree structure, and a default used in any needed view. A particularly clean version of this would require that its children really all have the same descendents, so that the semantics of the flagged node would be un-altered by the choice of subsequent classification.

A related problem concerns the organization of roots. In particular, it will be noted that there is no top-down structure over the set of roots (e.g., in Figure 1(iii)). It should be remembered, however, that the various contexts (including the various roots) can indeed be browsed, not top-down but from the bottom up, from the point of view of a particular focal content. If one wants to guarantee a view of all roots one could introduce an artificial leaf, descendent from all roots, whose upward view by design is therefore a tree containing all those roots.

In the end, of course, if hierarchical partition is rare, and partial overlap and random interconnection is the rule, then neither trees nor multitrees will be appropriate, and one needs to move back in the direction of general DAGs for representation, paying the corresponding price in complicating the interface.

Viewing the whole structure

For some applications a view of a non-tree portion of a multitree (e.g., the entire multitree) may be desirable. This might arise in particular when building on an existing multitree and wanting to take fragments from different subtrees, or perhaps in some unusual browsing situations. The fact that multitrees are not generally planar presents a challenge, and the problem becomes similar to one of general DAG visualization and browsing. A 3-dimensional layout may be

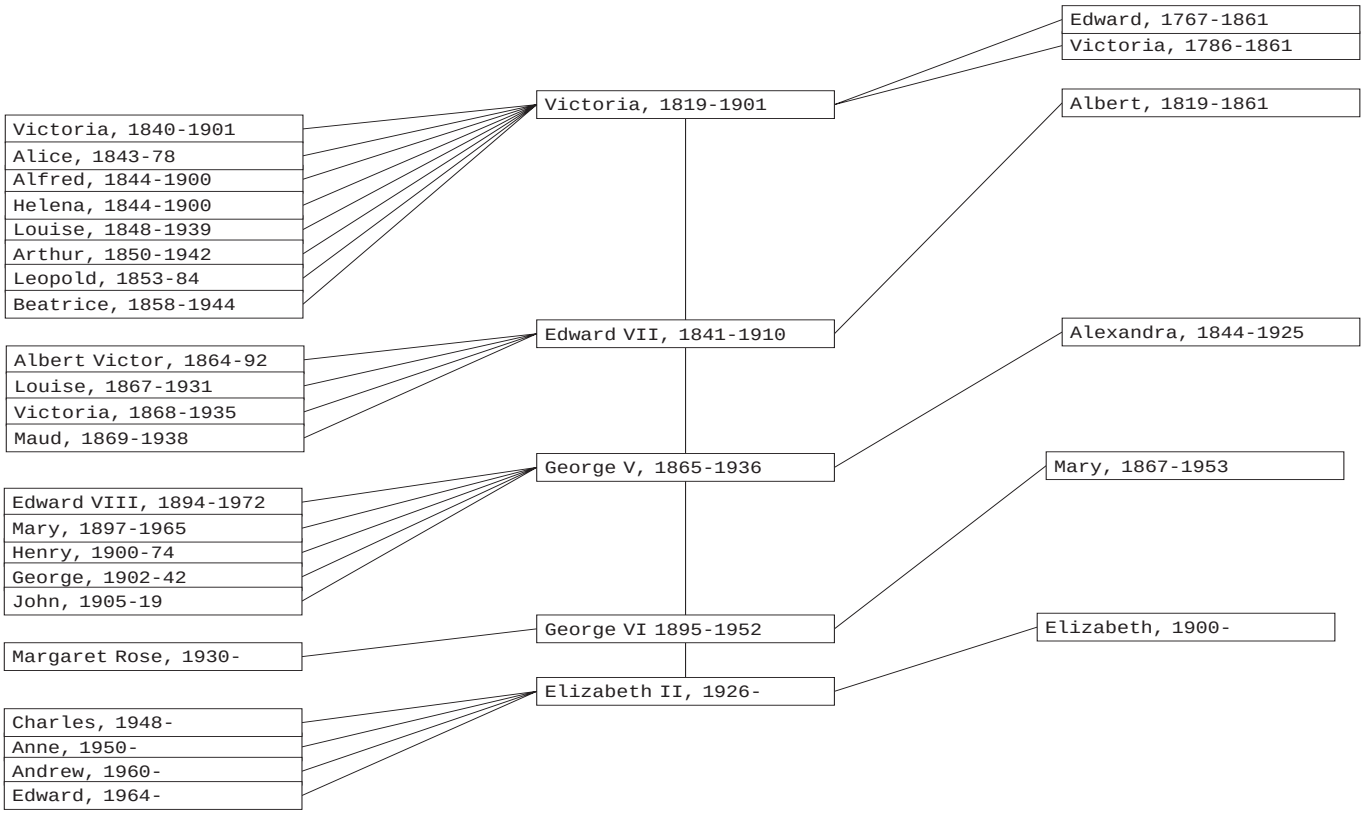

Figure 6 . Integrated fisheye view for a royal genealogy. This view shows both upward and downward fisheye views at once, i.e., the ancestral lineage (from Queen Victoria to Queen Elizabeth II), as well as both first order ancestors and first order descendants of that lineage.

best for this, with adjacent nodes placed near each other, and the direction of links preserved in the layout. Multitrees may allow special 3-D arrangements, since whole tree portions of interest may be assigned to their own planes in the structure.

Reuse out of context

There is a problem that always arises in reuse, no less for hierarchical structure and multitrees. That is, fragments composed in one specific context may not work well out of that context. This problem comes up in code reuse, where special discipline is needed to design and encapsulate modules so that they may be called from various contexts. In text systems fragments can contain anaphora, explicit reference to other sections, conceptual prerequisites, etc. all of which are resolved in a well written linear text. This has always caused problems for hypertext, where fragments may be encountered in various orders. The typical solution is to try to write fragments that stand alone, leading to an often jerky style.

For hierarchical reuse, the problem would be that a new tree assembled from fragments taken from elsewhere would not hang together well. The individual readings in a professor's course, just plucked out and glued together, might well lack establishing context and smooth transitions. There are two strategies however that should greatly ameliorate the problem. The first is to admit that some points in an existing structure might be more suitable candidates than others for reuse, and make the most use of those pieces that have less external dependency. The second is to note that new trees built in the structure can introduce newly created fragments. Thus it is quite possible to include new ad hoc fragments that can provide the segues and contexts for the old pieces that are being reused. A nice feature of this solution is also that these new fragments are automatically specifically associated with the new contextual organization in which the old fragments now appear.

Constructing Multitrees

Constructing multitrees, like building structure in general, presents a challenge. Perhaps one of the most reliable ways to construct good multitrees will be as in our initial example of professors setting up courses, i.e., by hand. It is intriguing, however to consider automatic methods that might try build multitrees from some pre-existing resources. For example, multitrees may be implicit in particular instances of more general graphs. Botafogo et al.[1] have presented an algorithm for finding trees within general hypertext graphs, based on the number of links into and out of a node (a root is defined to be a node that can reach every, or nearly every, node in the hypertext). The IGD system[4] allowed users to build such trees by hand on top of an existing DAG. If one could use such techniques to identify several trees within a hypertext, and then join them together, this would produce a multitree.

CONCLUSIONS

The principal contribution of multitrees is as a new coherent point on the structural continuum from trees to general graphs. Like all other points on that spectrum, they have their strengths and weaknesses, both in their graphical and semantic capabilities. As with DAGs in general, showing a whole multitree can be difficult, but large meaningful fragments can be shown as simple trees. Multitrees have natural notions of content and context, and seem appropriate for cases of reuse of hierarchical structure. Their appropriateness for more general representation (e.g., knowledge bases, or object oriented class systems) is an open question, but still they may provide a useful approximation just as trees sometimes do. For example Creech, et al.[2] have implemented a hypertext system for software reuse based on DAGs. It would be interesting to see if multitrees could serve as well. Even if multitrees are too constraining, it is possible that they can form a distinguished backbone, with other types of links serving to complete the structure. Thus the tree pieces can be used for some viewing and navigation purposes, and abandoned as needed.

Acknowledgments

We would like to thank Susan Dumais, Jakob Nielsen, and Scott Stornetta and Henri Weinberg for variously trying the browsers and reading drafts of this paper.

REFERENCES

[1] Botafogo, R.A., Rivlin, E., Schneiderman, B., Structural analysis of hypertext: identifying hierarchies and useful metrics, *ACM Transactions on Information Systems*, 10(2), 1992, 142-180.

[2] Creech, M. L., Freeze, D. F., Griss, M. L., Using hypertext in selecting reusable software components, *Hypertext'91Proceedings*, ACM, 25-38, 1991.

[3] Egan, D. E., Remde, J. R., Landauer, T. K., Lochbaum, C. L., and Gomez, L. M., Behavioral Evaluation and Analysis of a Hypertext Browser, *Proceedings of ACM CHI'89 Conference on Human Factors in Computing Systems* 205-210, 1989.

[4] Feiner, S. (1988). Seeing the forest for the trees: Hierarchical display of hypertext structure. *Proc. ACM Conf. Office Information Systems* (Palo Alto, CA, 23-25 March), 205-212.

[5] Furnas, G. W., The FISHEYE view: A new look at structured files. *Bell Laboratories Technical Memorandum*, 1982.

[6] Furnas, G. W., Generalized fisheye views. *Human Factors in Computing Systems CHI '86 Conference Proceedings*, Boston, April 13-17, 1986, 16-23.

[7] Landauer, T. K., Dumais, S. T., Gomez, L. M., & Furnas, G. W., Human factors in data access. *Bell System Technical Journal, (Special Issue on Data Bases)*, 61, 1982, pp. 2487-2509. Reprinted in: Journal of Information and Image Management. September, 1983, Vol 16(9), pp. 18-29.

[8] Parunak, H. Van Dyke, Hypermedia topologies and user navigation, *Hypertext'89 Proceedings*, ACM, 43-50, 1989.