Effectively View-Navigable Structures

George W. Furnas
Bellcore, 445 South Street
Morristown, NJ 07960-6438
(201) 829-4289
gwf@bellcore.com

Paper presented at the Human Computer Interaction Consortium Workshop HCIC'95
Snow Mountain Ranch, Colorado,
February 17, 1995

ABSTRACT: Certain fundamental properties are necessary for informations structure to be effectively view-navigable as they get larger. View-navigation here means the familiar method of moving from one locally visible part of the information structure to another until a target is found. Fundamental properties needed include: views of the structure must be small, connecting paths must not be too long, and the paths to targets must be discoverable by a Navigator. This paper explores concepts useful in understanding these constraints, including that of the viewing graph of a structure, as well as to-sets and residue.

I. INTRODUCTION

The World Wide Web (WWW) is the most recent popular example of a vast online information world. All who have use it are impressed by the range and richness of the content accessible simply by view navigation, i.e., wandering around and clicking on things they see on the screen. Soon after users are also typically struck by how hard it is to find anything specific by such navigation. In these initial stages of the evolution of the WWW, its structure has not been supportive of any form of targeted navigational search.

The fundamental problem with structures like the Web is that they are big and the users resources are small. If the computer interface and the human eye and brain allowed one to see and comprehend everything in the structure at once, there would be no problem in picking out a desired target directly. And on the other hand, if one had all the time in the universe to look around, wandering around tediously to find things would be less of an issue. Unfortunately, one can only see and comprehend a little bit of a large structure at once, and interactions must lead to success sooner rather than later.

Thus the efficient use of the classical resources of space and time, often investigated formally in computer science, are critical to shaping effective view navigation, particularly as information structures get very large. This paper will examine the role of these two resources in view navigation, and how their efficient use puts constraints on what kinds of information structures are effectively view navigable. It will then examine the implications for the design of navigable information systems and interfaces to them.

The basic framework

We will be considering systems with static structure over which users must navigate to explore and find things, e.g., lists, trees, planes, grids,... The structure contains elements of some sort (items in a list, nodes in a hypertext graph) which are organized in some logical structure. We assume that the interface given to the user is navigational in the following sense: First, the user at any given time is "at" some node in the structure, i.e., has some special capabilities and view specific to that node (e.g., a view of the local neighborhood). For example, for a list the user might have window centered on a particular item, or for a hypertext, a view of the current page with its links referring to other pages. Second, given the view the user has of part of the structure as seen from the current location, the user has an ability to move to anything else in that view. Thus in a list the user could click on an item at the bottom of the window and that item would

scroll up to the middle of the window. In a hypertext web, tue user could follow one of the visible links in the current hypertext page. *View navigation* is this iterative process of viewing, choosing and moving around -- until a target is found.

It is useful to distinguish the simpler *View Traversal* component of view navigation. View traversal is simply the "mechanical" process of moving around iteratively by clicking on things in the each view. This is the style of interaction called a Direct Walk by Card, et al [3]. We choose the terminology "view traversal" here to emphasize the view aspect, since we wish to study the use of spatial resources viewing implies. *View Navigation* differs from the more rudimentary view traversal in that it additionally involves the ability to actually *find* good paths to targets. I.e., not just that is possible to click on things to get places, but that it is possible to discern the right things to click on to get to desired places.

This paper will consider Traversability and Navigability in order, with the goal of understanding the implied resource problems as structures get very large. We first investigate Efficient View Traversability (EVT), fundamentally noting that we at least want it to be mechanically possible to reach things with the efficient use of time and space. This is a basic need, because if no good view traversal exists from a starting point to a target, it makes the navigational problem of finding it moot. On the other hand we do need Navigability(NAV), i.e., we need finding a good path to be possible and resource efficient. So the basic requirements for NAV will be discussed. Finally we look at when we have both EVT and NAV: Effective View Navigability (EVN).

II. EFFICIENT VIEW TRAVERSIBILITY: EVT

We begin, then, with basic requirements for efficient view traversal: allowing good mechanics for moving around the information structure by viewing, selecting and moving. We set forth very minimal conditions; one might ask much more of an information structure (in fact we will later for navigability), but as information structures encompass thousands, millions, even billions of items, if these traversability conditions are not met, the systems will be impossible to use.

Definitions and Fundamental Requirements

We assume that the elements of the information structure (the items in a list, nodes in a hypertext graph, etc.) are organized in a logical structure that can be characterized by a *logical structure graph*, connecting elements to their logical neighbors. For a list this would just be line graph, with each item connected to the items which proceeds it and follow it. For a hypertext the logical structure graph would be some sort of web. We make no assumption regarding the logical graph whether or not all or some of its edges are directed or undirected. We do assume the logical graph is connected and finite.

We define the v*iewing-graph* (actually a directed graph) of an interface to a logical structure as follows. It has a vertex for each vertex in the logical structure. A directed edge goes from a vertex *i* to vertex *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the view-digraph might be identical to the logical graph, but need not be. It is possible to include in the current view, points that are logically far away (e.g., the root, home page, top of the list). Conversely it may also exclude edges to nodes that are logically close by, e.g., if there are too many to view at once.

We are interested in classes of viewing-graphs that behave well even as the structures get very large.

"Behave well" here means "allows the efficient use of space and time resources during view traversal." The structure is large, but users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view navigation these limitations translate correspondingly into two requirements on the viewing graph. First, since the structure is assumed to be huge, and the screen is small, the individual views of the structure must be substantially smaller than the structure. Certainly it is assumed to be impossible to see the whole structure in is entire scope and detail; only a subset of the complete and detailed structure may be seen from any given point, *x*. In the viewing graph, the vertices in that visible subset are pointed to by the outgoing edges from *x*. In graph theory, the number of outgoing edges is called the "out-degree", hence we have our first requirement:

Requirement EVT1 (small views). The out-degree of vertices in the viewing graph must be "small" compared to the size of the structure.

The second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable amount of time to get from one place to another. We want to be able to get anyplace in a "small" number of view navigation steps. This means that the shortest path between any vertices *x* and *y* must not be too long. Since the length of that path (in number of edges) is the graph distance, we say

Requirement EVT2 (short paths). The distance between pairs of vertices in the viewing graph must be "small" compared to the size of the structure.

We will say that a viewing graph is Efficiently View Traversable (EVT) insofar as it meets both of these requirements.

A few words are in order about the notion of size in these requirements. The size that really matters in practice is dictated by the resources available. If a window only has room for 50 items in a list, then "small" means within that limit. Despite this reality, for much of the discussion here we will talk in coarser terms, following complexity notation -- how well things scale in a general mathematical sense -- because it tells more generally how much you could improve things by investing a certain amount more of resources (e.g., how much better of would you be if you could redesign to see 60 items in the list).

Thus we will want to talk about identifiable classes of structures, each of variable size, *n*. We will say that the quantities of interest (out-degrees for EVT1 or distances for EVT2) are small insofar as they grow more slowly than the size of the structure, e.g., less than linearly.

Also note that while "small outdegree" is a property of a single vertex, and "small distance" is a property of a single pair of vertices, the requirements EVT1&2 relate to the set of all such quantities in the whole viewing graph. For example, the statement that "out-degrees must be small" could mean, "usually small", "small on average", "small even in the worst case." In this paper we will usually use a worst case analysis, limiting the maximal out-degree (MOD) in EVT1, and maximal pairwise distance (or "diameter", DIA) in EVT2. *1

We define the following function *EVT(G)*, that tells for any given graph, *G*, how well it satisfies requirements EVT1 and EVT2:

Definition. For a viewing graph *G*, we define *EVT(G)=(MOD(G), DIA(G))*, the ordered pair of maximal outdegree and diameter of the graph respectively.

We will use this function to compare the view traversability of various classes of information structures.

Having defined these basic requirements we turn to a simple example.

>> EXAMPLE 1: A SCROLLING LIST

Consider an ordered list, sketched in Figure 1(a). Its logical structure connects each item with the item just before and just after it in the list. Thus the logical graph (b) is a line graph. A standard viewer for a long list is a simple scrolling window (c), which when centered on one line (marked by the star), shows a few lines on either side. Thus a piece of the viewing graph, shown in (d), has edges going from the starred node to

*1 One could consider average-case, or median case, etc. At other times, one might want to consider the whole set of out-degrees or distances for graph, defining $EVT_F(G)$ to give the whole viewability and distance functions, i.e., $EVT_F(G) = (v,d)$ where v and d are functions. Letting $K = \{1,..., |Edges(G)|\}$ then $v: K \rightarrow I_{0+}$ with v(i) = degree of vertex i, and $d: KxK \rightarrow I_{0+}$ with d(i,j) = interpoint distance between vertices i and j. EVT as defined in body of the paper just gives the maximum values of the corresponding two component EVT_F function

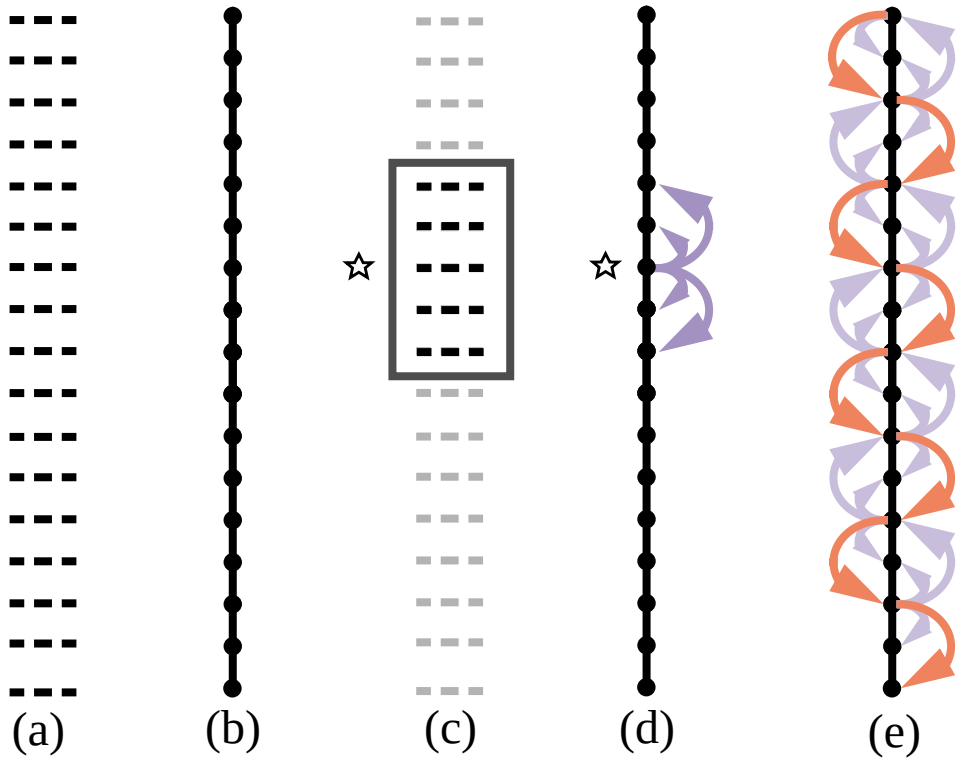

Figure 1. *(a) schematic of an ordered list (b) logical graph of the list, (c) local view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) diameter of viewing graph is O(n).*

all the nearby nodes that you can see in the view. The complete viewing graph would have this replicated for all nodes in the logical graph.

The first thing to note is that this viewing graph satisfies the first requirement EVT1 nicely. Regardless of the length of the list, the out-degree of the viewing graph is small: the view size is the small fixed size of the viewing window. The diameter requirement of EVT2, on the other hand, is in trouble. View traversal for the scrolling list happens by clicking on a line in the viewing window, e.g., the bottom line. That line would then appear in the center of the screen, and repeated clicks could move all the way through the list. As seen in (e), moving from one end of the list to the other requires a number of steps linear in the size of the list. This means that

$$EVT(\text{ SCROLLING-LIST}_n\ ) = (\ O(1), O(n)\ ),$$

and overall it is not very Effectively View Traversable. This is presumably consistent with most people's intuition that while individual views in a scrolling list interface are reasonable, such a scrolling technique would be a miserable interaction technique for even moderately large lists of up to a few hundred items (where scrollbars were a needed invention), and impossible for huge ones (e.g., billions, where even scroll bars will fail).

Design for EVT

The scrolling list of Figure 1 is not very effectively view traversable. If the list gets very large, it takes many view-navigation steps to move from one end to the other, thereby failing on the short-paths (diameter) requirement. There are structures, however that meet the EVT requirements much more satisfactorily. One goal of the research here is to bring an understanding of good (EVT) structures to bear on the design of information navigation systems. The approach is first to find nice classes of structures that have the desired distance and degree properties and understand how can be (or perhaps already are) useful as view-

traversable information structures. A second goal is to try to figure out how to fix situations where logical structures do not have the desired properties, and so EVT strategies are not immediately available. One might for example look at how to build EVT viewing graphs from non-EVT logical graphs, or how to attached fully preformed EVT graphs onto non-EVT logical graphs, etc. A few small examples of this approach are sketched next.

Some Efficiently View Traversable Structures

We begin by considering two example graphs that have good EVT performance. These are particularly simple examples in that the viewing graph is essentially identical to the logical graph, i.e., they are based on classes of graphs that have the proper degree and diameter properties.

>> EXAMPLE 2: LOCAL VIEWING OF A BALANCED TREE.

Trees are commonly used to represent information, from organizational hierarchies, to library classification systems, to menu systems. An idealized version appears in Figure 2(a) . This is idealized in that it is

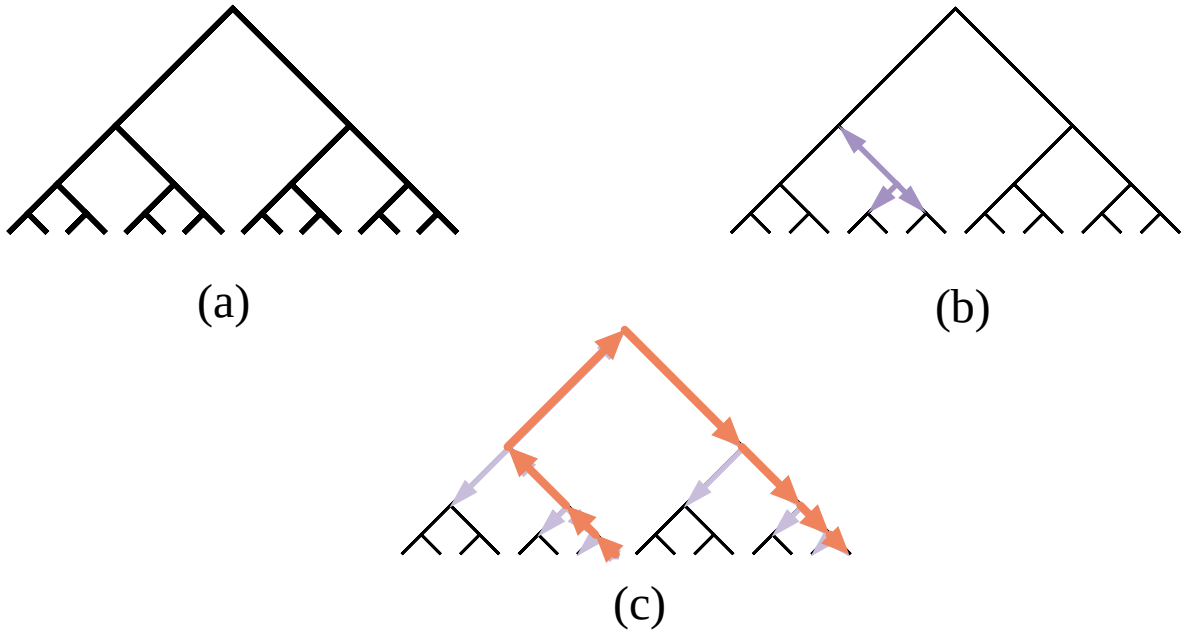

Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) diameter of viewing graph is O(log(n)).*

regular (same branching factor everywhere) and balanced (descendant subtrees from any vertex are of comparable size). A typical tree viewing strategy makes visible from a given vertex its parent and its children (b), i.e., the viewing structure essentially mirrors the logical structure. Here, regardless of the total size, *n*, of the tree, the outdegree of each vertex in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied Requirement EVT1. The diameter of the balanced tree is also well behaved, being twice the depth of the tree, which is logarithmic in the size of the tree, and hence *O(log n)*. Thus

$$EVT(\text{BALANCED-REGULAR-TREE}_n) = (O(1), O(\log n)) \quad {}^{*2}$$

>> EXAMPLE 3: LOCAL VIEWING OF A HYPERCUBE.

This example considers the EVT of a hypercube. In information contexts, a *k*-dimensional hypercube might be used to represent the structure of a factorially designed set of objects, where there are *k* binary

factors. For example objects are either big or small, red or green, round or square, in all various combinations. Figure 2(a) diagrams the logical structure of a three dimensional example. In the logical graph, edges connect objects that differ by only one feature. In a view navigation paradigm, one could imagine a viewing strategy that mirrored the logical structure: showing the logical neighbors of the current item.

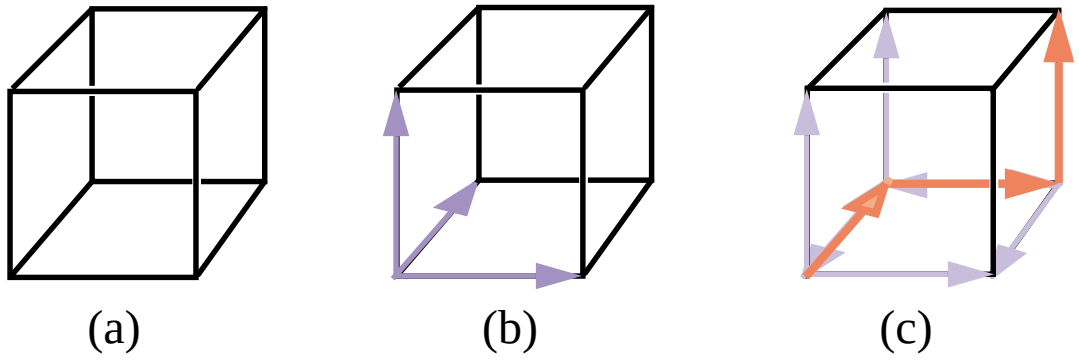

Figure 3. *Another Efficiently View Traversable Structure: (a) logical graph of a "hypercube" (here just a 3-cube), (b) part of the viewing graph for giving local views of the hypercube showing the outdegree O(log(n)), (c) diameter of viewing graph is also O(log(n)).*

In a *k*-dimensional hypercube, each vertex has *k* neighbors (one for each feature that could vary). There are in total $n=2^k$ vertices. Thus the view size is logarithmic in the size of the whole structure. The diameter is also *k* (at most *k* features would have to be changed to move from any object to any other), so we have

$$EVT(\text{HYPERCUBE}_n) = (O(\log n), O(\log n)).$$

The conclusion of these two examples is simply that, if one can structure the world into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths are sufficient for very large structures. Knowing a large arsenal of such structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

It is all well and good that nicely EVT structures exist, but what can one do if saddled with an information structure whose logical structure does not directly translate into a viewing strategy that is EVT? The beauty of separating out the viewing graph from the logical graph is that while the domain may dictate the logical graph, the interface designer can craft the viewing graph. Thus we next consider a number of strategies for improving the EVT behavior by the design of good viewing graphs. We begin by showing several ways to improve the view navigation of lists, then to some ways to slightly improve even tree view navigation. Then we finish up this section of Efficient View Traversability by mentioning some general, if weak, results, and some observations.

>> Example 4: Fixing the List (version 1) - Going Multidimensional

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multicolumn list (Figure 4).The logical graph is the same (a), but by folding as in (b) one can give views that show two dimensional local neighborhoods (c). These local neighborhoods are of constant size,

---

*2 Another version of balanced trees would have constant diameter: make a tree of fixed depth but increasing branching factor. The size of a balanced tree is $n=b^d$, for branching factor, *b*, and depth, *d*. This means, as the example in the main body illustrates, that $d = log(n)/log(b)$ and so for fixed branching factor depth is logarithmic. But it also it means $b = n^{1/d}$, so one could get constant depth if one allows the branching factor, and hence the view (out-degree) to grow as *d*-th root of the size. For even moderate depths, say *d*=10, this would do quite well. In this version EVT= $(O(n^{1/d}), O(1))$.

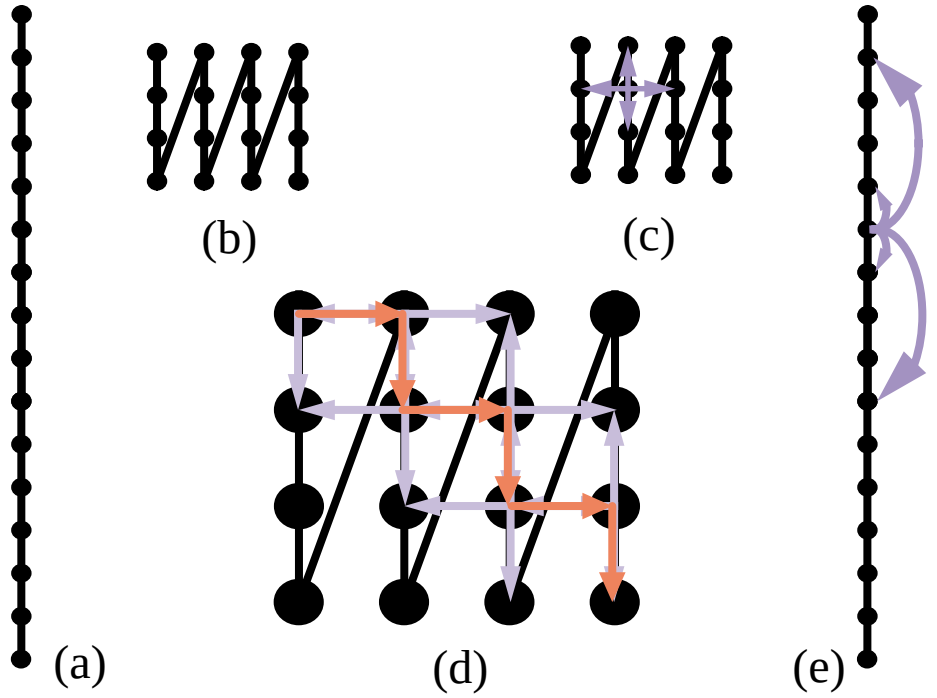

Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) the viewing graph shows the 2-D neighbors, yielding out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) the viewing graph portion of (c) unfolded illustrating how things farther away are shown to decrease diameter.*

regardless of the size of the list, so the outdegree of the viewing graph is still constant, but the diameter of the graph is now sublinear, being related to the square-root of the length of the list, so we have

$$EVT(\text{MULTI-COLUMN-LIST}_n) = (O(1), O(sqrt(n)).$$

This square-root reduction in diameter presumably explains why it is common practice to lay out lists of moderate size in multiple columns (e.g., the “ls” command in UNIX, a page of the phone book). Note that the total printed area on paper taken up by a list is not substantially different if it is single or multiple column -- a long one-column rectangle and the short fat multicolumn one will have comparable area. The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another. Note also that lists that are too short do not need the multiple columns, *O(n)* does not hurt yet. On the other hand lists that are huge would not benefit enough, since then *O(sqrt(n))* is not compressive enough, so lists are often abandoned altogether for large collections.

Note also that really long lists, such as those in a full phone book are folded up in 3-D: multiple columns per page, and pages stacked one upon another. We take this format for granted, but imagine if one only had 2D to place all those numbers in (or, heaven forbid, 1-D). In fact, the layout of any printed text in books in general makes use of this same advantage of multiple dimensions to decrease the movement required to get from one arbitrary place to another. Now the world is trying to use hypertext to link pages together non-dimensionally. As we structure them, we want to be careful to at least to do better than the stacked-in-3-D arrangement of books!

One way to think about what has happened in 2-D is illustrated in Figure 4 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list. The critical thing to note is that some of the

edges of the viewing graph now point to vertices that are not local in the logical structure of the graph. This is a very general class of strategies for decreasing the diameter of the viewing graph, further illustrated in the next example.

>> EXAMPLE 5: FIXING THE LIST (VERSION 2) - FISHEYE LIST SAMPLING

It is possible to use non-local viewing edges to improve even further the view-traversability of a list. Figure 5 shows a viewing strategy where the vertices included in a view are spaced in a geometric series. That

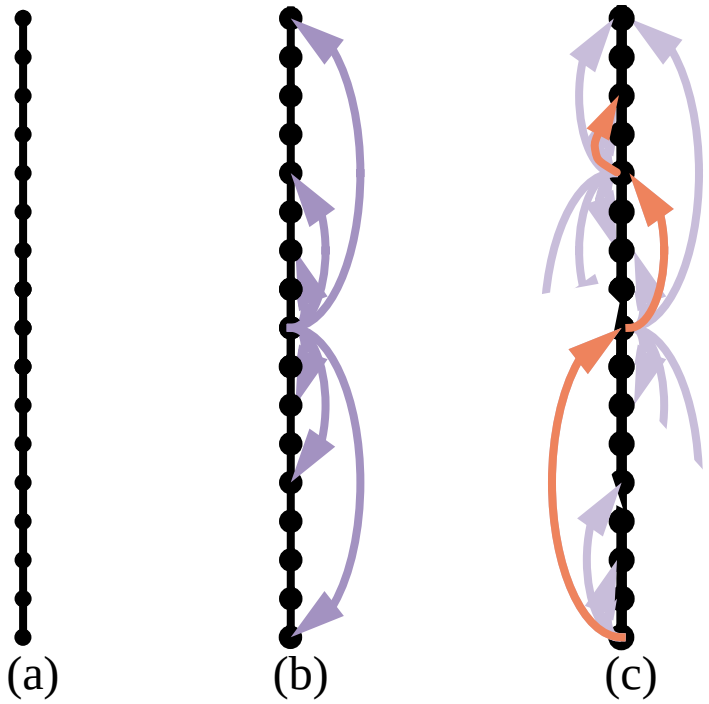

Figure 5. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fisheye sampled list, showing that out degree is O(log(n)), (c) diameter of viewing graph is now O(log(n)).*

is, from the current node, one can see the nodes that are a distance 1 away, 2 away, 4 away, 8 away, 16 away, etc. This sampling pattern might be called a kind of fisheye sampling, in that it shows the local part of the list in most detail, and further regions in successively less detail.

This strategy results in a view size that is logarithmic in the size of the list. Moving from one node to another ends up to be a process much like a binary search, and gives a diameter of the viewing graph that is also logarithmic. Thus

$$EVT(\ \text{FISHEYE-SAMPLED-LIST}_n\ ) = (\ O(\log n), O(\log n)\ ).$$

This is the same as the EVT of a hypercube, and it turns out that the two are very closely related.[*3]

So lesson from these first two examples is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance edges. In the future it would be valuable to know more examples, and more systematic strategies.

---

*3 The resemblance is cleanest for the circular version of a list that is some power of 2 in length: First the elements of the list are numbered in binary. Then in the fisheye-sampled strategy the first item of the list will be connected to its neighbors in the hypercube taking the bits of the binary address as hypercube dimensions. Considering this and its various symmetrical versions at other nodes yields an interesting subgraph of the hypercube with similar EVT properties.

>> Example 6: Fixing the List (version 3) - Tree Augmentation

Another sort of strategy involves not just adding new (long-distance) edges, but adding other vertices to the viewing graph that are not in the original logical structure. This allows various shortcut paths to share structure, and reduce the total number of edges needed, and hence the general outdegree. Here we show an even more specific, but still quite general strategy: gluing a structure known to be EVT onto a given non-EVT structure. In this case a tree is glued onto the side of the list. Traversal is done predominantly through the tree. Thus in Figure 2, a bunch of new vertices are introduced (*O(n)*), and viewing edges are intro-

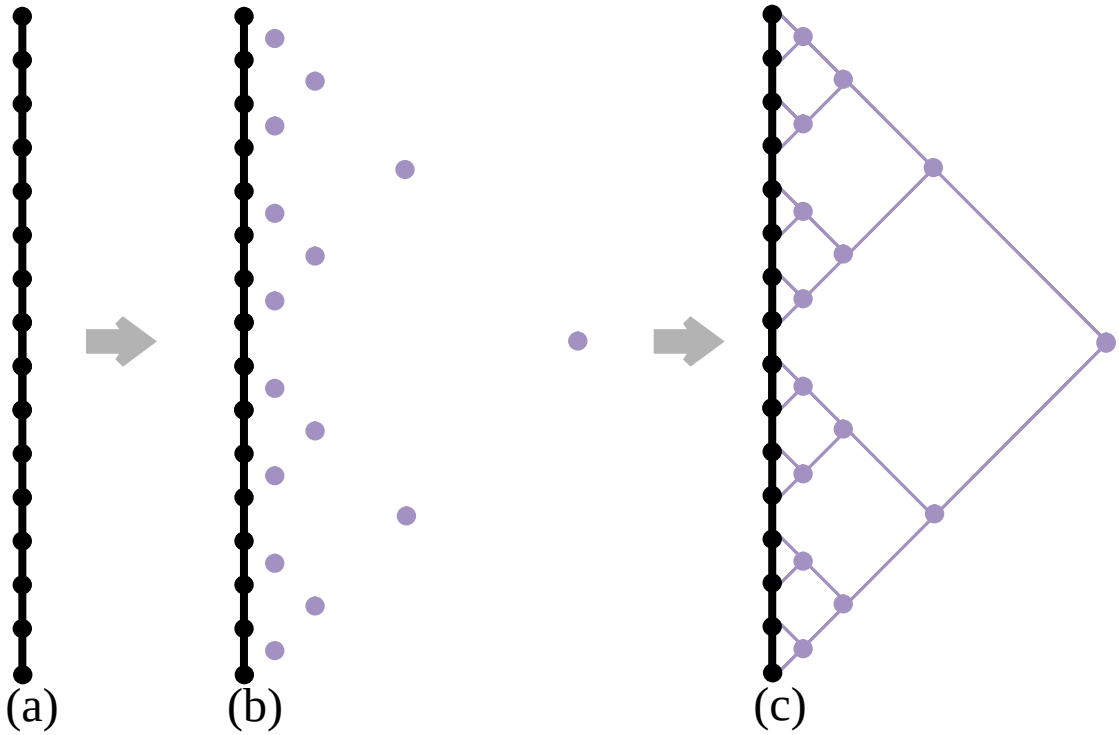

Figure 6. *Improving EVT of a 1D structure by adding a tree. The original line graph (a) is augmented by copies at successively lower resolution (b). These new vertices are locally interconnected in the viewing graph to their neighbors at the same scale and to close points at the neighboring scales. The resulting structure has constant viewsize but logarithmic diameter*

duced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original vertices by going through the tree, we get

$$EVT(\text{TREE-AUGMENTED-LIST}_n) = (O(1), O(\log n)).$$

>> Example 7: Fixing the List (version 4) - Pan&Zoom

The final example of fixing a list is in many ways similar to tree augmentation. It also introduces new vertices, an essentially identical number of them, but it connects them more richly. The interconnections introduced are in fact a superset of those in the tree augmentation strategy. The result essentially implements a discrete pan and zoom of a window over the one dimensional structure. The list is replicated by successively lower resolution copies (copies that sample only every second, every fourth, every eighth... list element.). The viewing graph connects vertices to their neighbors at the same scale and to certain close points at neighboring scales, as shown in Figure 7 and Figure 8.

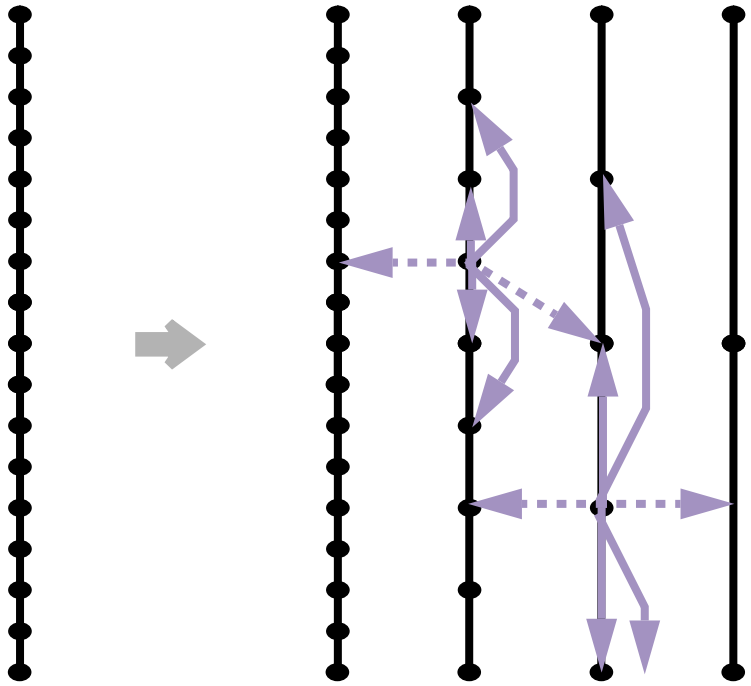

Figure 7. *Improving EVT of a 1D structure using discrete zoom. The resulting structure has a fixed viewsize and diameter O(log(n)).*

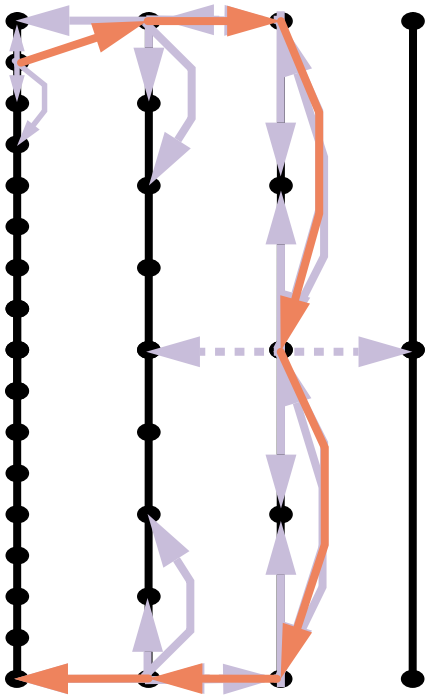

Figure 8. *Improving EVT of a 1D structure using zoom (cont). The resulting structure has a fixed viewsize and diameter O(log(n)).*

Moving between layers amounts to a discrete zoom, and moving within a layer is a pan.[*4] The viewsize and diameter of the structure are now comparable to that of a tree augmentation, with slightly better aver-

age distance, slightly worse out-degree, but still,

$$EVT(\text{PAN-ZOOM-LIST}_n) = (O(1), O(\log n)).$$

Thus one reason that a zoom capability is good for an information interface is that it improves the EVT of the structure.*5

>> EXAMPLE 8: IMPROVING THE TREE

One last set of examples of how to improve traversability shows that even good graphs like trees can be improved, or more specifically, one can improve the diameter with small cost to the outdegree. These improvements, in some cases, do not change the O() analysis, merely altering constants, but as mention in the initial discussion of "size" in real interfaces, constants can be important, and so we mention the strategies here. Two are shown in Figure 9. The first simply includes the root in every view. This strategy is

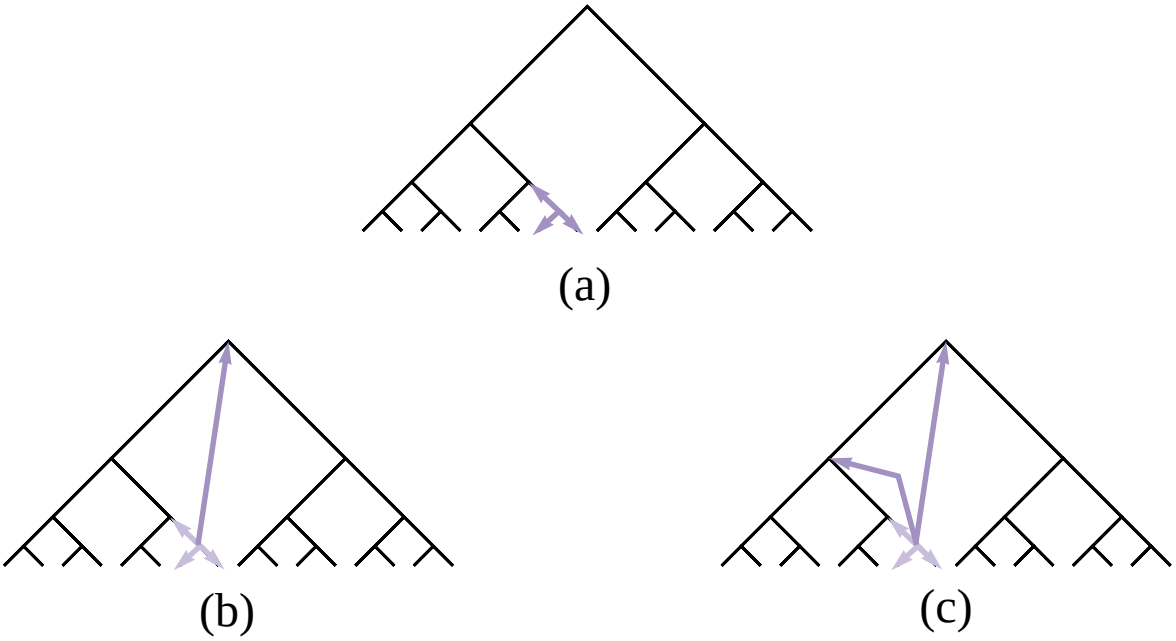

Figure 9. *Ways to improve a tree. (a) The logical graph and with a part of the purely local viewing graph. (b) Always including the root in the view decreases the worst-case paths (diameter) by half. (c) Including the ancestral lineage makes the view O(log(n)), but decreases the average path length by about half.*

common in many hierarchical information access systems (e.g., menu hierarchy systems). According to the EVT analysis here, a principal virtue is that it decreases the diameter of the graph by essentially a factor of 2 (you can always jump back to the root in one step to get to the far side of the tree), yet increases the outdegree by only one. The second strategy shown in the figure is a bit less common. It amounts to showing the ancestral lineage of any vertex in addition to the local neighborhood. This not only decreases the length of paths through the root (e.g., the worst case paths and hence the diameter), but decreases most other paths as well. The path between two vertices in a tree always goes through their lowest common ancestor. The pictured strategy makes it possible to get to that ancestor in one jump, decreasing the path

*4 The only wrinkle here is that zooming (especially zooming out) might not naturally be thought of as a view navigation step: the zoomed-out duplicate of a node is not separately visible item in the view. Thus we have been calling the viewing graph here becomes some generalized "immediate access" graph, with edges pointing to those other vertices that can be easily accessed from the current vertex. This immediate-access-graph generalization of the viewing graph is probably worth pursuing in future work.

*5 Furnas and Bederson analyze the continuous 2-D version of this in [5].

length about half between essentially all pairs. Note however, that this actually increases the outdegree from *O(1)* to *O(log(n))*, which means it might not be worth it for extremely deep trees or very limited view space.

The strategy of Figure 2(c) is basically a 0-order fisheye view [4]. It is a general characteristic of fisheye views that, in addition to showing local structure, they show a *few* aspects of the logical structure that are far away, thereby typically decreasing diameter, at relatively low cost to view size.

Conclusions and Observations about Efficient View Traversability

Conclusions about fixing viewing graphs

The previous examples have shown that adding edges, or adding on another complete structure, can sometimes yield better EVT performance. One might wonder if anything very coherent can be said about such operations in general. Consider the operation of adding or subtracting edges. Adding edges to a graph never lengthens any existing paths in the graph, so the good news is that if a graph has small diameter (EVT2) then after adding edges it will still have small diameter. Obviously also, if the graph has small views (EVT1), then subtracting edges will only decrease view sizes, so it will still have small views. The bad news is the converse is not true. So adding and subtracting edges does not have simple effect on EVT: one can wreck the EVT1 while fixing EVT2 and vice versa. Thus the trick will be, if the diameter is too big, to add edges and not overwhelm views, or if the views are too big, to subtract edges and not increase diameter.

In achieving this balance it is useful to note that if a graph has too big a diameter, the impact of selectively adding edges can be much greater on decreasing diameter than on increasing view size, to net positive effect. A sufficient way to illustrate this point is in terms of combining structures. Suppose we have a graph G_1 that we want to fix, because EVT(G_1) is poor simply because of long paths. Suppose also that we have another graph G_2 over at least those same vertices but with different edge interconnection patterns, such that EVT(G_2) is good. Then it is easy to show that the union of these two graphs will have reasonably good EVT. This follows from the proposition below.

First note that we say EVT of one graph is less than or equal to that of another, $EVT(G_1) \leq EVT(G_2)$, iff both its outdegree and diameter are less than or equal to those of the other. Then we can state,

Proposition (EVT-union). Let $G_1 = \langle V_1, E_1 \rangle$, $G_2 = \langle V_2, E_2 \rangle$, $G_3 = \langle V_3, E_3 \rangle = \langle V_1 \cup V_2, E_1 \cup E_2 \rangle$ *and* $V_1 \cap V_2 \neq \emptyset$. Further, let $EVT(G_1) = (v_1, d_1)$ and $EVT(G_2) = (v_2, d_2)$. Then,

$$EVT(G_3) \leq (v_1+v_2, d_1+d_2)$$

Furthermore, if $V_1 \subseteq V_2$, *then* $EVT(G_3) \leq (v_1+v_2, d_2)$

proof. The proof of this proposition is trivial. Both outdegree results follow from the fact that in forming the union graph the edges leaving any vertex will be the union of the edges leaving it from both graphs (if it exists in both), and hence the number in the union will not exceed the sum of the numbers in the separate graphs. This is turn will be bounded by the some of their worst cases v_1+v_2. The first diameter result follows from the fact that, since the graphs overlap, the distance between any two vertices cannot possibly be greater than the sum of the two diameters. The second diameter result comes from the fact that the union graph, G_3, has just the vertices of G_2, so for any pair of vertices, the shortest path in G_2 still exists, so their distance in G_3 must be at least that small. QED.

Corollary. For any graph of size n in the class Γ_1, we associate another graph of size $f(n)$ in the class Γ_2, such that $V_1(n) \subseteq V_2(f(n))$. Now let Γ_3 be the class of graphs formed by the union of corresponding elements, i.e., $\Gamma_3(n) = \Gamma_1(n) \cup \Gamma_2(f(n))$. Now, if $EVT(\Gamma_1(n)) = (O(v_1(n)), O(d_1(n)))$ and $EVT(\Gamma_2(n)) = (O(v_2(n)), O(d_2(n)))$, we have

$$EVT(\Gamma_3(n)) \leq (max(O(v_1(n)), O(v_2(f(n)))), O(d_2(f(n))))$$

proof. This follows directly from part 2 of the proposition and the fact that:

$$O(v_1(n)+v_2(n)) \leq O(max\ (2v_1(n), 2v_2(n)) \leq O(max\ (v_1(n), v_2(n))).$$

Example 6 illustrated the corollary: $\Gamma_1(n)$ was the list of length n, $\Gamma_2(2n-1)$ was the tree whose leaves were all the nodes of the list, hence $V_1 \subseteq V_2$. The EVT of their union was guaranteed to be no worse than *(O(1), O(log n)).*

This formalizes the point that you fix a viewing graph with bad diameter properties by gluing view traversable one onto it everywhere. The result is the original structure augmented by an easily traversable backbone. The new graph will have good diameter, without dramatic damage to the viewsize.

As a taxonomic note, the tree augmented list example suggests a special class of EVT repairs, what might be called "external" solutions. Define an EVT viewing graph G_2 over a graph G_1 to be *external* if $V_1 \leq V_2$ and $E_2 \cap E_1 = \varnothing$. I.e., the EVT backbone structure is purely auxiliary, using none of the logical edges; it introduces purely new nodes and edges to ensure EVT. For example one could put an auxiliary superstructural hierarchy over a set of web pages (like YAHOO, in the current WWW).

EVT Jump&Show

Efficiently view traversable structures have an additional property useful for human information-structure interactions. Suppose that one has been examining the structure at some vertex *A* and that, by some sort of auxiliary mechanism other than view traversal, has made a jump to vertex *B*, for example as the result of a search. It might in general be useful for the user to see how the current position *B* relates to the original position, e.g., how one might have gotten there by a view traversal.

EVT implies that an arbitrary jump has a small rendering of the view traversal. The move from *A* to *B* is guaranteed to be possible in a small number of steps, each of which comprises a small view. Since the union of a small number of small things cannot be too big, it follows that such a union view of the whole view traversal process is reasonably small. In particular $|\bigcup \text{views}| \leq \text{DIA} * \text{MOD}$. This view-traversal can be rendered dynamically, as in a movie, or statically, as in a map, and in either case is guaranteed to be small. In fact, in some way, EVT, by balancing time and space requirements for moving around is essentially about trying to make structures that have traversals that can be conveyed with limited resources, e.g., a small number of total bits. A similar concept was explored in some detail for continuous Pan&Zoom in [5].

III. NAVIGABILITY

So far we have been considering attributes of large information structures that make them friendly for view traversal. Such concerns are important, but do not go far enough. It does one little good if a short path exists but one does not know what that path is. The structure must also be *view navigable.* That is, it must be possible to figure out the path from one's current position to the desired target. Thus we next discuss how it might be possible to figure out the next step in a short path to the target from looking at the current view. In order to choose which next step to take, there must be some information associated with each outlink at the current node, and some procedure which, using that information and a target specification, will let a navigator correctly choose the next step in a good path to the target.

Definitions and Fundamental Requirements

Here too we begin with some definitions and requirements. First, critical to the discussion will be the notion of *specifiers*. Specifiers (specs) are data structures of some sort that special operators use to do useful things related to set operations, such as testing membership. Individual nodes have specifiers, and sets of nodes can have specifiers.

Of the special operators for specifiers, one is the *match* operator. It "matches" the specifier of a node against that of a set and return true *(t)* if the specified node is a member of the specified set, and false (*nil*)

otherwise. For example, a node specifier might just be some unique arbitrary ID number. Then a set specifier might be an enumeration, e.g., list of ID numbers. The match operator will take an ID number and check the list for that number, returning t iff it finds it. Alternatively, the node specifier might be a unique set of attributes that the node possesses. A set specifier would be some arbitrary set of attributes, and would specify the set of nodes having that attribute. The set of all things specified by a specifier (i.e., the set of all things whose specifiers match it) are its *extension.* Thus a node specifier has as its extension the associated unique node, and a set specifier has as its extension the set of all nodes whose specifiers match it.

Another special operator of importance is a *quasi-unifier.* Where matching is used to test a single item's specifier against a set's, a quasi-unifier takes two set specifiers is determines if the two corresponding sets intersect. (We call it a quasi-unifier here in that a standard unifier also usually is required to return a specificier for the intersection set. That specifier is not important to this analysis.)

Now we are prepared for further definitions of this

> *Definition.* The *outlink-info, OLI(i,j)* , associated with the *j*th outlink of node *i* is a specifier associated with that outlink.

The outlink-info is used to by the navigator to decide whether to follow that link. The prototypical example of outlink information, of course, is a link label. In this analysis, that label is taken to specify something, as we will discuss shortly.

The target for a search is also taken to be a specifier. For the purposes here, we will have it be an individual node-spec, i.e., the specifier of the target node, but the theoretical framework here should support analysis of incompletely specified targets, or target sets In that case the target-specifier is a set-specifier, and matching operations turn into quasi-unifications (presumably with consequent complication).

In the current real world, an information navigator is typically a human, but in the future it might also be an automatic agent sent out to navigate the information structure. For the purposes here we will simply abstract it as follows:

> *Definition.* The *navigator* is a process or procedure that moves from node to node as follows. Given the specification, *S*, of a target node, *T*, the navigator applies the function, *Nav(OLI(i, j), S) ≡ Match(OLI(i, j), S)*, to each outlink *j* in *i*. The navigator randomly chooses between the outlinks that return true. It stops at the vertex whose spec is *S*.[*6]

That is, the navigator matches the target node spec against all the outlink specs and follows one that matches (see Figure 10). (Note that this version of the navigator has no history, or other extra information. Such extensions are areas for future research.)

Now, in order to describe when view Navigation is possible we will need the following definition.

> *Definition.* The *true-to-set*, or simply *to-set*, written *to-set(i, j)*, of a link, *j*, at node *i* is the set of vertices which that link "leads to".

For the purposes here, we will interpret "leads to" to mean "leads to via a shortest path". The basic concept, however, is that the to-set is the set of things one should following the corresponding link to get to.

---

*6 This *Nav* function is more general than it might at first seem. In particular, one might think that a more general *Nav* function would be of the form *Nav(OLI(i,•), S, j)*, i.e., that the decision about outlink *j* could not be made in isolation, rather depending on all the other outlink info at node *i* (hence "*OLI(i,•)*"). For example, one might have $Nav_1 \equiv Match_1(OLI(i,j),S)$ return a real number, a goodness of match number, not just *t*/*nil*, for each link. Then *Nav()* would be a function which returns *t* for *j*'s that are maxima of Nav_1. This can be encompassed in the current framework, however, by encapsulating any such dependencies and folding them redundantly into the individual outlink-info. I.e., if necessary the full set of other link specs from the naive, interdependent version are packaged along with each link spec to create a new self-contained, if larger, link-spec. The only reason this may be a problem is it changes both the size of the outlink-info, and the general character of the link-specification language. Those consequences should be explored.

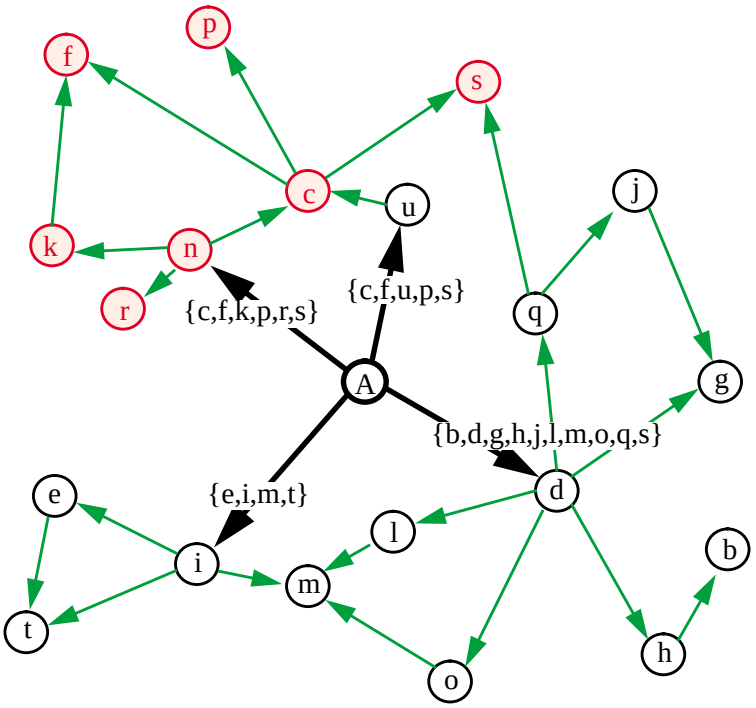

Figure 11.*The sets listed in curlybraces are the to-sets associated with links from A. These are the vertices that the given out-edge leads to via a shortest path.*

(See Figure 11). One can imagine trying to consider other notions of "leads to", such as just "via a reasonably good path".

Definition. The *inferred-to-set*, *inferred-to-set(i,j)* of link *j* is just the extension of the outlink info specifier, *OLI(i,j)*, i.e., the set of all target nodes whose specs would match *OLI(i,j)*.

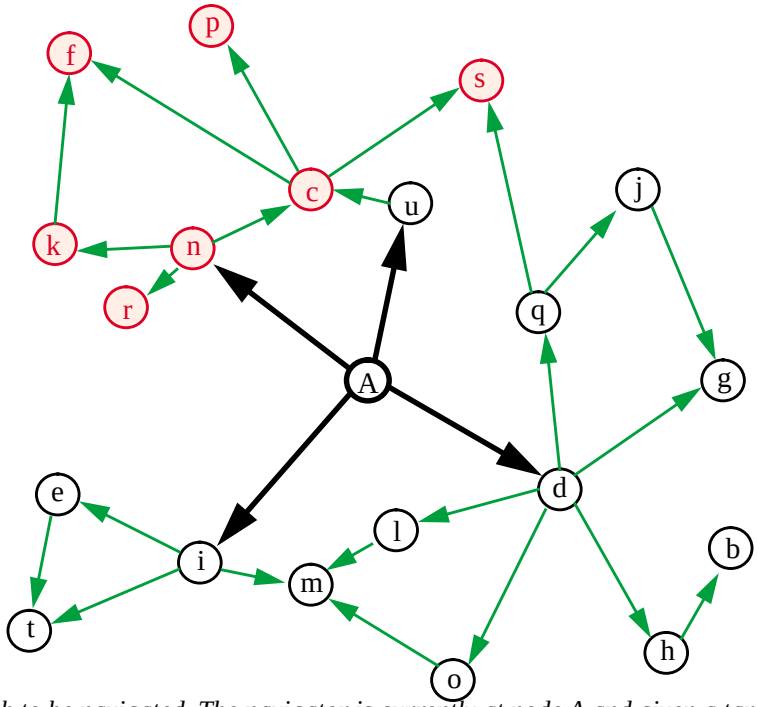

Figure 10. *A graph to be navigated. The navigator is currently at node A and given a target, will evaluate Nav() on for each of its four outlinks and follow one of the links that returns t.*

Definition. The outlink info of edge *j* of node *i* is said, for a target *T*, to be *NotMisleading-wrt-T*, *NoMis(i,j,T)*, iff *T* being in the *inferred-to-set(i,j)* implies *T* is in the *true-to-set(i,j)*.

Intuitively, "if the label says *T* is this way, it's this way". Note that the converse is not being required here; the label need not be complete, underspecifying its *true-to-set*.

Definition. The outlink info of node *i* is said to be *well-matched-wrt-T* (*T≠i*) iff

1. (NoMis condition) *NoMis(i,j,T)* for all outlinks, *j, and*

2. (Coverage condition) *T* is in the inferred-to-set of at least one outgoing edge of *i*.

Definition. The outlink info of node *i* is said to be *well-matched* iff it is *well-matched-wrt-T* for all *T≠i*

The idea here is that if the outlink info is well-matched, the navigator will not be lead astray by any outlink-info, and at least one link's outlink info will guide toward the target. (See Figure 12)

Now we are prepared to discuss the following rather simple requirement for navigability:

Requirement VN1(navigability): The outlink-info must be everywhere well matched.

We need this requirement because of the following proposition:

Proposition: The navigator is guaranteed to find good paths to targets iff the outlink info is everywhere well-matched (i.e., requirement VN1 is met).

Proof. Consider the navigator starting at some arbitrary vertex *A* and seeking vertex *T*.

• The forward direction: Assume the outlink info is everywhere well matched. Let the navigator be at any arbitrary node in the graph. If it is at *T* it is done. If the navigator is one step away

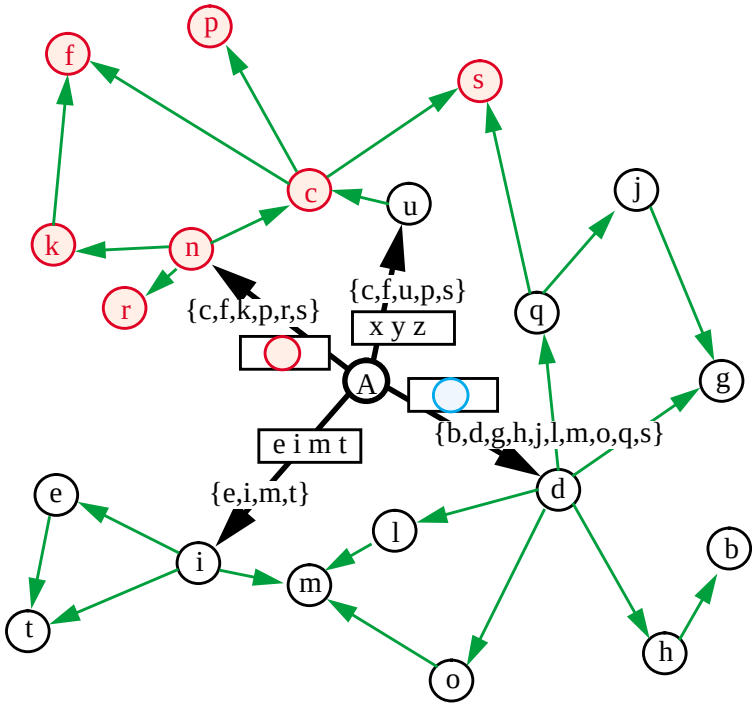

Figure 12. *Link info associated with links from A is in the boxes. The link-info on the lower left is an enumeration, and in the upper left is a feature: a red circle. These are both well matched. The link info for links on the right is not-well matched. The residue of f at A is the red-circle label. The residue of e at A is its appearance in the enumeration label. The node h has no good-residue at A*

from its target, then outlinks leading directly to *T* will have *T* in their to-sets. If the outlink-info is well-matched, then *Nav()* will return *t* for any such edges, and the navigator will take one such outlink and reach the target via this one-step, hence shortest, path and be done. Otherwise, consider those neighbor nodes of the current position which are next on some shortest path to *T*. By definition, *T* will be in the true-to-set of the outlinks connected to those nodes. If the outlink-info is well-matched, then *Nav()* will return *t* for just those edges, and there will be at least one. The navigator will follow one of them and be one step closer to the target. Since the graph is finite, it will eventually reach the target.

• The reverse direction: Given that the navigator can start at any *A* and find a shortest path to any *T* using *Nav()* as prescribed, we want to show that this implies requirement VN1.

If the navigator can start at any *A* and find a shortest path to any *T*, it can certainly take a first step from *A*, when looking for any *T* not equal to *A*. It takes such a step because *Nav()* must have returned true for some outlink from *A*, and therefore *T* is in the inferred-to-set of some outlink of *A*, so condition 2 (coverage) of well-matchedness is met for *A* wrt *T*.

Now since that first step is by assumption on a shortest path from *A* to *T*, it follows that *T* is in the true-to-set of that edge of *A*, and hence that edge is *NoMis-wrt-T*. On repeated starts from *A* the navigator will eventually follow all outlinks for which *Nav()* returns true, i.e., all those for which *T* is in the inferred-to-set, and each such is therefore *NoMis-wrt-T*. Since other edges are *NoMis-wrt-T* by default (they do not match *T* at all), we conclude that all the edges of *A* are *NoMis-wrt-T* and we conclude that *A* is well-matched-wrt-T. Since this logic holds for arbitrary targets $T \neq A$, the node *A* is well-matched. Since it holds for all starting places, *A*, we conclude that the universal success of the navigator implies that outlink-info is everywhere well-matched.

QED.

The critical observation in all this for designing information systems is that, to be navigable, the outlink-info of a link must in some sense describe not just the next node, but the whole to-set. This is a problem in many hypertext systems: Their link-labels label adjacent nodes and do not give a sufficient indication of what else lies beyond. In a sense, for navigation purposes, “highway signage” might be a better metaphor for good outlink-info than “label”. The information has to convey what is off in that direction, off along that route, rather than just where it will get to next.

This analysis so far has been in terms of how the world looks to a navigator that successively follows outlinks using outlink-info until it gets to its target. The navigator wants a world in which it can find its target. Now let us think about the situation from the other side: how the world looks from the perspective of a target, T. The target wants a world in which it can be found. This complementary perspective brings up the important notion of *residue*.

Definition. The *residue* of a target *T* at node *i* outlink *j*, written *Res(T, i, j)*, is (1) the outlink-info of that link if the to-set of that outlink contains *T* or (2) *nil*, otherwise. If the outlink-info for that outlink is *NoMis-wrt-T*, and the residue is non-*nil*, it is called *good residue* of *T*.

That is, there is good residue of *T* if the outlink info will help the navigator find *T*. (Refer back to the caption of Figure 12).

Note that in order to be findable by navigation from anywhere in the structure, the target must have good residue at every node. I.e., in order to be able to find *T* you have to be able have some sniff, some residue, of it, no matter where you are, to get you started chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of Requirement VN1:

Requirement VN1a (residue distribution): Every node must have good residue at every other node.

This is a daunting challenge. There are numerous examples of real world information structures without good residue distribution. Consider the WWW. You want to find some target T from your current location, but do not have a clue of how to navigate there because your target has no good-residue here. There is no trace of it in the current view. This is a fundamental reason why the WWW is not navigable. For another example consider pan&zoom interfaces to information worlds, like PAD^{++}[6]. If you zoom out too far, your target can disappear entirely leaving no residue, and you cannot find it again.

The requirement VN1a is far from trivial, basically because of the following scaling constraint.

Requirement VN2. Outlink-info must be "small".

To understand this requirement and its impact, consider that one way to get perfect matching or equivalently perfect universal residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each edge (i.e., each edge is labeled by the complete set of things it "leads to", as in the label of the lower left outlink from vertex *A* in Figure 10). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exists each node. (Cf., the assumptions of the Bellman Ford algorithm for packet routing). This "enumeration" strategy is presumably not feasible for view navigation since it does not scale. It is *O(n)* at each node, *O(n^2)* info over whole structure. The labeled views must be much smaller than the size of the structure. We need to reduce the outlink-info, e.g., to *O(log(n))* or *O(1)* at each node; the outlink-info must somehow *specify* its to-set in some way more efficiently than by enumeration.

A critical conclusion to draw is that outlink-info is a scarce resource. Targets must share residue; residue must be reused.

The issues underlying good residue distribution are intriguing and complex. The notions of specifications and the way they interoperate with each other (a matter of their *intensions*) and with the sets they specify (a matter of their *extensions*) are at the heart semantics. Only a few modest observations will be listed here.

Before doing so, however, a few methodological comments are in order. This section so far has concerned establishing some first principles. It is worth sketching a future research program that at this point might be useful: Study in some detail examples of any structures where navigation seems to work. i.e., where requirements VN1 and VN2 appear to be satisfied (e.g., taxonomic trees, certain dimensional structures, and addressed hypercubes); then try to induce and then verify general principles and strategies for satisfying the VN requirements; finally try to apply those principles elsewhere for design.

For now, we simply return to the following assorted observations.

Discussion and Observations on View Navigability

• One may view Requirement VN2 as saying that the minimum description length of the to-sets must be small compared to the size of those sets. That is equivalent to saying that the sets are not random. The bad news is that this raises the spectre of absolutely non-navigable structures, a few examples of which will be given shortly. The good news is that meaningful information worlds will presumably not involve random structures. The question is, where in the middle do our information worlds lie -- how navigable can they be?

• It follows from the definitions that, if the graph is connected as we have been assuming, *T* is in at least one to-set for each node. If the graph is navigable then *T* is in at least one inferred-to-set for each node, i.e., *T* must match at least *n* specifiers (one for each node in structure). If we could argue that most of these have to be different, it would seem to say something about the richness of the object semantics of a navigable domain, since T must be a member of lots of semantically specifiable sets.

• In addition to understanding the nature of the individual to-sets that must be specified for view navigation, one might also want to know how they vary as one moves from node to node in the structure. It is easy to say something about how the to-sets of adjacent nodes look. Consider the local part of the structure shown in Figure 13. (Here the outdegree of vertices *x* and *y* are just taken to be three, but the result is trivi-

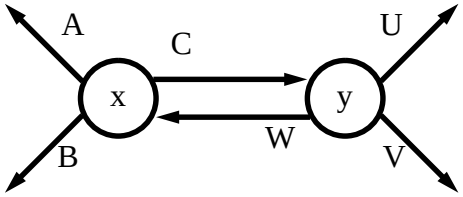

Figure 13. *Two adjacent nodes, x and y, in a structure, with to-sets associated with each outlink*

ally generalized.) It is easy to show that if *x* has an outlink to *y* as in the figure, then $C \subseteq \{y\} \cup U \cup V$. And of course symmetrically, if y has a link to *x*, then $W \subseteq \{x\} \cup A \cup B$. In both of these relationships, the left hand side might be a strict subset if there are vertices that can be reached equally well both ways. Then moving from *x* to *y* will change which way is shortest. We can also conclude that if *x* and *y* are doubly linked, $A \cup B \cup \{x\} - W = U \cup V \cup \{y\} - C = M$ exactly that set of borderline vertices. This result says that in fact these sets are fairly tightly coupled, and so their specifiers, and presumably semantics must also be tightly coupled. A fabric of sets and corresponding semantics must be woven.

A special example is in a tree, where there is always only one path to any vertex, so M is necessarily empty. The to-set of the link leading into node is partitioned among to-sets of the (non-returning) links leading out of it. To obtain view navigability, those to-set partitions must be reflected in the link labels. This is most familiarly accomplished when the tree is rooted to form a hierarchy. Then the to-sets moving from the root form successive partitions and view navigability is obtained by labeling those links with category labels that semantically carve up the sets correspondingly. Note that if the leaves are scrambled or in some other way the to-sets do not match the category labels, navigability is destroyed.

The tough research question here is to understand what other structures and webs of semantic labelings can be made to mirror each other satisfactorily.

• Similarity-based navigation. One interesting class of unforgettable structures makes use of similarity both to organize the structure and run the navigator. Objects are at nodes, and there is some notion of similarity between objects. The outlink-info of a link is simply a copy of the object at other end of link. The navigator knows or can compute the similarity between objects and chooses the outlink whose associated object is most similar to its ultimate target.

The success of the similarity approach presumably requires that nodes for similar objects be linked in the structure. This is not sufficient, however. As Example 10, discussed in detail a bit later will illustrate, it is possible to take any graph structure and find a set of artificial objects whose similarity structure is isomorphic to it (and hence can be laid out so that similar things are linked), yet the structure is completely non-navigable. This is essentially because all similarity has been designed to be purely local in an important sense, and so there is no good-residue of things far away.

In similarity-based navigation, it is interesting to note that the inferred-to-set of a link is the set of all objects that are more similar to that neighbor than to any others. One interesting question concerns what exactly is the nature of the residue of a target. The answer to this somehow requires further analysis of the notion of similarity. There are many theories of similarity (Tversky, Shepard,...), but most are based on notions of features, attributes or dimensions on which objects share values to greater or lesser extents.

Such an approach is used in constructing the non-navigable examples below, and which will be instructive in understanding certain classes of navigable structures in the future.

• Intrinsically Non-Navigable sets. It is worth noting that some sets of objects are intrinsically non navigable. It helps in the understanding of navigability in the abstract, and also raises the point that insofar as the real world is like such sets, it means that designing navigable systems is going to be hard.

>> EXAMPLE 9: NON-NAVIGABLE SET 1 - COMPLETELY UNRELATED THINGS.

A set of *n* completely unrelated things is intrinsically not navigable. For example just consider an abstract alphabet of size *n*. All subsets are information full, with no structure or redundancy, and hence irreducible. Thus there is no structure whose outlinks can be labeled by anything other than enumeration. One might complain that this example is not interesting since there is no intrinsic structure at all. Hence we consider the next example.

>> EXAMPLE 10: NON-NAVIGABLE SET 2 - LOCALLY RELATED STRUCTURE.

In this example we create sets of *n* only "locally related" things. For example a "chain, " can be constructed as follows. Take a large alphabet of size *n* (like that used in the previous example), and generate a random permutation of it, e.g.,

JPBXLGVOYCFKWUQAHZREDIMTNS

Then create as "objects" successive pairs from this ordering. The logical structure is dictated by the overlap pattern in the pairs:

JP-PB-BX-XL-LG-GV-...-MT-TN-NS

Despite the fact that "similar" objects are adjacent in this organization, there is no way to know how to get from, say, LG to anything other than its immediate neighbors There is no residue of things far away. [*7]This case generalizes to any graph structure with m edges: lay an m-alphabet randomly on edges of the graph. Objects, placed at the vertices, are union of edge-letters. Overlap, i.e., similarity, is isomorphic to the graph, but the structure is still non-navigable because there is no constraint on what lies more than one link away. There is no way to label the outlinks to reflect the to-sets.

There is no better way to structure the graph, others just start to reduce it to the first non-navigable case

IV. Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is view navigable, then both the mechanical constraint of EVT on diameter and outdegree and the residue constraints of VN must hold. In this section we make some informal observations about how the two sets of constraints interact.

• The combination of EVT and VN implies that on average set specifiers have to specify reasonably large sets. Since by assumption everything can be reached from a given node, the union of the to-sets at each node form the whole set of *n* items in the structure. If there are *v* edges leaving the node, the average size of the to-set is *n/v.* If the structure satisfies EVT2, then *v* is small compared to *n*, so the average to-set is quite large.

VN1 requires that outlink info faithfully specify these sets. This would seem to imply something about the ubiquity of high level semantics in navigable structures. If we assume the specs are related to the semantics of objects, and that specs with large cardinality extensions are high level semantics, it follows that

*7 This example is akin to what Tversky has offered as a model of a "qualitative dimension". But this is only one example of a non-quantitative similarity dimension, one that is non-navigable. One might try to design higher order chains, where features were shared more distantly, producing a more navigable qualitative dimension.

high level semantics play a dominant role in the structure. This deserves closer analysis -- how are large sets specified? If, for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. One would need semantic operators on specifiers to create new specifiers that specified large to-sets. The equivalent of a union operator would not help much, since “small” cardinality unions of “small” sets do not make large sets. A complement operator could help some, since the complement of a set small set is large. That would suggest that if mostly only small sets are naturally described, the liberal use of some sort of negation operator on specifiers might be common in making navigable structures.

Consider taxonomic trees, for example. Upward links often have large cardinality to-sets, and are typically not specified by Natural Kinds categories. The link from “bird” up to “animal” leads, really, not just to other animals but everything else in the taxonomy that is “not bird”.

• Earlier it was noted that the to-sets of a structure form a kind of overlapping mosaic which, by VN1 must be mirrored in the outlink info. There are also presumably semantic consequences of enforcing the diameter requirement of EVT2. In particular, the neighboring to-sets have to differ more radically under EVT2. Consider by contrast the to-sets of the line graph, a graph that does not satisfy EVT1. There the to-sets change only very slowly as one moves along, with only one item being eliminated at a time. This suggests a generalization based on the following:

> *Proposition.* Given a starting node *A* in a graph, any other node *T* is in the intersection of all the to-sets on any shortest path, *S,* from *A* to *T.* Further, if *T* is locally maximally remote from *A,* i.e., no neighbor of *T* is further from *A,* then, *T* is the *unique* element in the intersection of all the to-sets on any shortest path, *S,* from *A* to *T.*
>
> *Proof.* First, note that for any node *B* on the path *S,* its outgoing edge along *S* must be on a shortest path from *B* to *T.* Thus by definition, *T* is in the to-set of that edge. Since this is true for all such *B, T* is in all the to-sets along the way, and hence in their intersection. To see that a if *T* is locally maximally distant from *A,* then it is the unique element in the intersection, assume to the contrary that there is distinct other node *T’* in the intersection. Then by the definition of to-sets, the shortest path from *A* to *T* is also part of a shortest path from A to *T*’. Since *T’* cannot between *A* and *T,* and is distinct from *T,* it must be beyond *T.* Hence there is a node on the shortest path from *T* to *T’* that is adjacent to *T* and further from *A,* violating the assumption that *T* was locally maximal. Thus no such *T’* can exist, and *T* is unique.

The implication of this proposition under the further constraints EVT2, i.e, that the length of the path must be short compared to the size of the structure, is that the to-sets must be able to whittle down the structure in a small number of intersections. Given that proposition result must hold for every A, such a quick whittling constraint must play a not insignificant role in an EVT structure. This, for example, is what makes a balanced binary tree EVT2, but a very unbalanced one not EVT2. In terms of view navigation, this constraint must also be mirrored in the semantics used for link-info -- the semantics of the domain must carve it up efficiently.

V. General Discussion EVT, VN and EVN

The material in this working paper is at best only the beginning. However, it is hoped that various concept of view traversability and view navigation provide a useful framework for thinking about some of the problems of interactions with increasingly large information worlds.

In view traversal, it is important to think explicitly about how things will scale, and to look further at how view traversal can be enhanced, by linking to things further away, or by adding on special navigational exostructure.

In view navigation we must pay more attention to residue management. It is important to note that the distribution of good residue is a global property. Residue is a global resource. In a distributed information environment like the WWW you do not have control over the structure and cannot enforce the residue distribution properties. If one has a new page to put on the web, how is it possible to ensure that it will have good residue throughout the rest of the network so people can find it. Who shall manage the global distribution of residue?

It is also clear that long-range and big picture aspects are important for navigating big structures. Large tosets must often be specified, and residues of remote vertices must be brought into views, both for diameter and navigability reasons.

One assumption throughout this work is that the structure and its outlink info is static. An important way to gain flexibility that may help the residue problem is to allow dynamic outlink info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

As a final comment, it is useful to remember some closely related classes of applications for this sort of analysis. Most notable is social routing of information: some information or question comes in to me. I know I'm not the right recipient for it so I pass it on to someone I know who might have a better idea of what to do with it, then pass it on in turn as necessary. In this case nodes are people and edges connect those who talk to each other. There is a navigator process stationed at each node and its the message that gets routed around, not the navigator.

Acknowledgment

This work was supported in part by ARPA grant N66001-94-C-6039. The authors would like to thank Will Hill and Maria Slowiaczek for their very helpful comments on drafts of this working paper.

References

1. Bederson, B. B. and Hollan, J. D., Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proceedings of ACM UIST'94*, (Marina Del Ray, CA, 1994), ACM Press, pp 17-26.
2. Botofago, R. A., Rivlin, E., and Schneiderman, B. Structural analysis of hypertexts: identifying hierarchies and useful metrics. ACM Transactions on Information Systems, 1992, 10(2), pp 142-180.
3. Card, S. K., Pirolli, P., and Mackinlay, J. D., The cost-of-knowledge characteristic function: display evaluation for direct-walk dynamic information visualizations. In *Proceedings of CHI'94 Human Factors in Computing Systems* (Boston, MA, April 1994), ACM press, pp. 238-244.
4. Furnas, G.W., Generalized fisheye views. In *Proceedings of CHI'86 Human Factors in Computing Systems* (Boston, MA, April 1986), ACM press, pp. 16-23.
5. Furnas, G. W., and Bederson, B., Space-Scale Diagrams: Understanding Multiscale Interfaces. In *Human Factors in Computing Systems, CHI'95 Conference Proceedings* (ACM), Denver, Colorado, May 8-11, 1995, 234-201.
6. Perlin, K. and Fox, D., Pad: An Alternative Approach to the Computer Interface. In *Proceedings of ACM SigGraph `93* (Anaheim, CA), 1993, pp. 57-64.