To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

Requirement EVT1 (small views). The number of out-going links, or out-degree, of nodes in the viewing graph must be "small" compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

Requirement EVT2 (short paths). The distance (in number of links) between pairs of nodes 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. There are many "reasonable senses" one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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

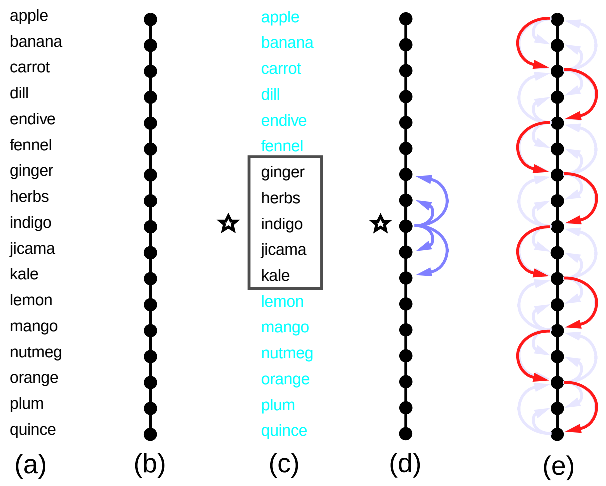


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),[3]$$

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically "in the limit proportional to *1*", i.e., constant -- an excellent score. *O(n)* means "in the limit proportional to *n*"-- a pretty poor score. *O(log n)* would mean "in the limit proportional to *log n*"-- quite respectable.

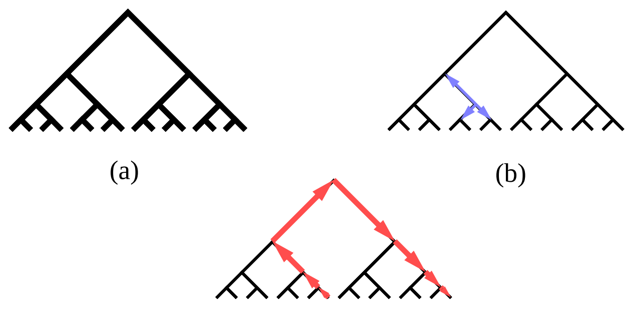


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

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

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4] One way to think about what has happened is

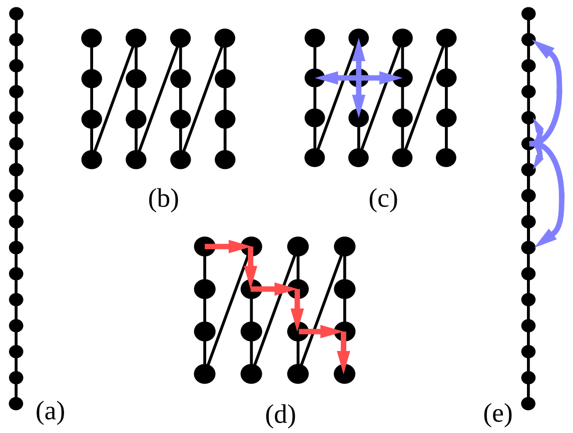


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

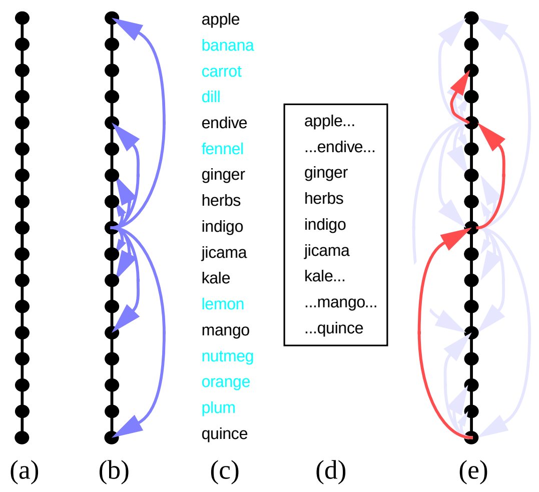


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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

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

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed,

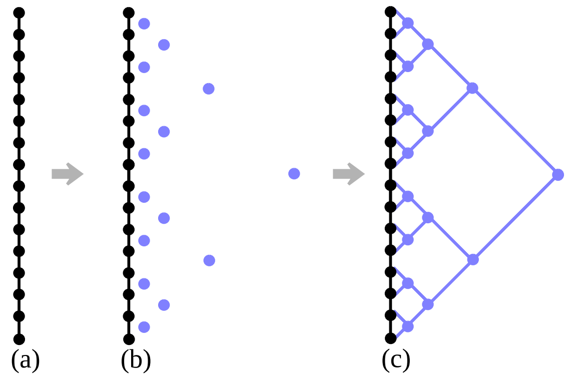


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

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

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n).* [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

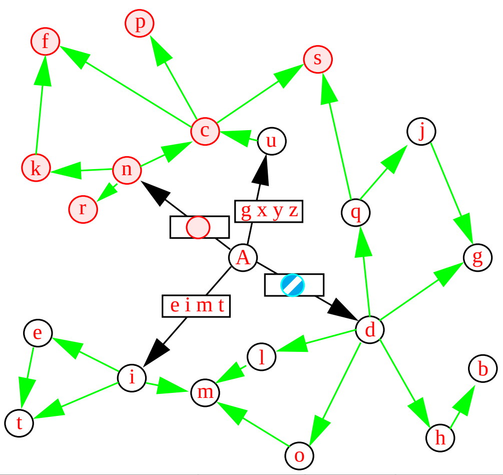
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently "leads to". If the navigator's target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	▨
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set.*)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it “leads to”, as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This “enumeration” strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled “The Rest”, which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

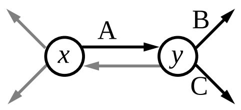


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to "BrownThings" and "LargeThings" but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

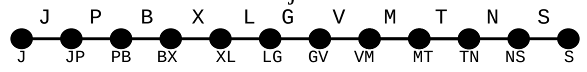
Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy. All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make "objects" at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label "the rest" can be used for some links, but this can be shown not to rescue the situation.

graph, including the sphere that the bloodhounds can see (smell?) into the "viewed" neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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* links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious "the rest" labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link's to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG's, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI'91 Proceedings,* 1991, 189-194.

To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

Requirement EVT1 (small views). The number of out-going links, or out-degree, of nodes in the viewing graph must be "small" compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

Requirement EVT2 (short paths). The distance (in number of links) between pairs of nodes 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. There are many "reasonable senses" one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

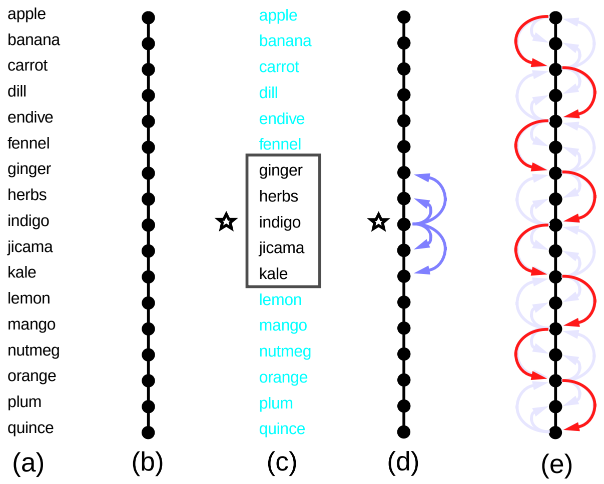


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),^3$$

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically "in the limit proportional to *1*", i.e., constant -- an excellent score. *O(n)* means "in the limit proportional to *n*"-- a pretty poor score. *O(log n)* would mean "in the limit proportional to *log n*"-- quite respectable.

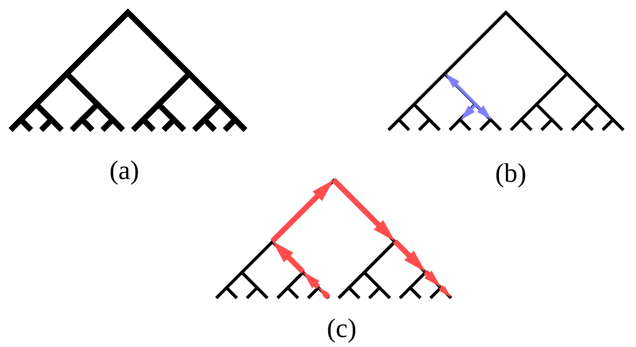


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

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

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4] One way to think about what has happened is

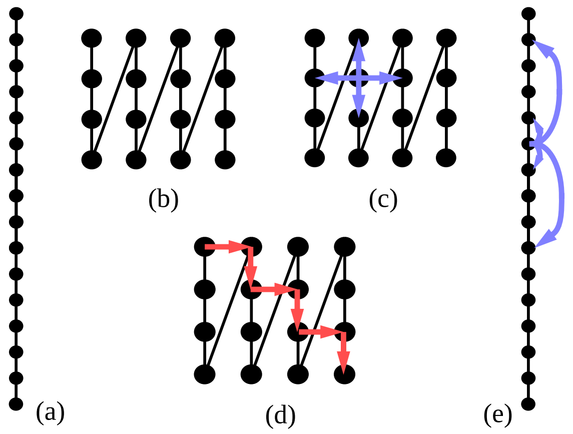


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

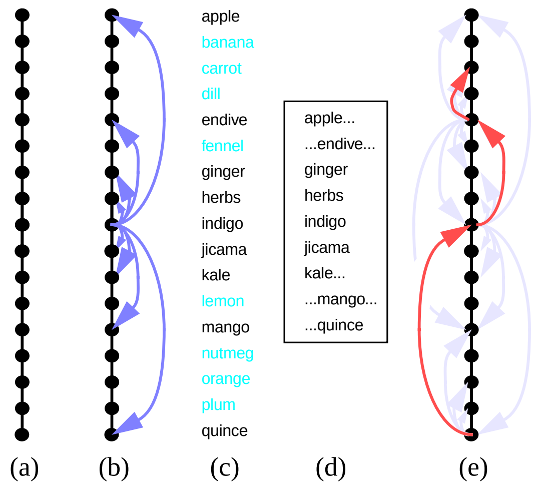


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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

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

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed,

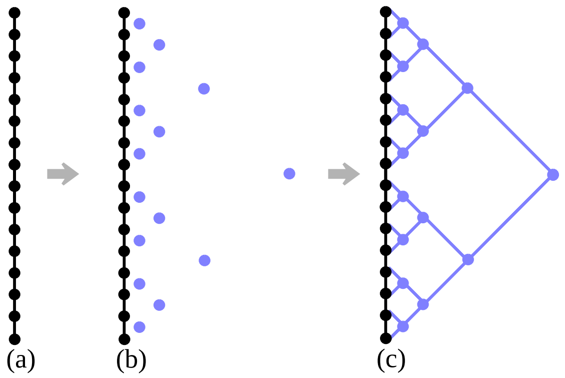


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

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

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n).* [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

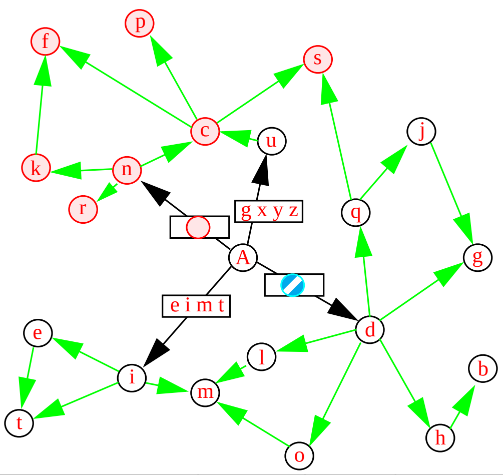
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently “leads to”. If the navigator’s target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	▨
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set.*)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it “leads to”, as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This “enumeration” strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled “The Rest”, which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

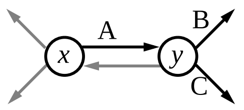


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to "BrownThings" and "LargeThings" but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

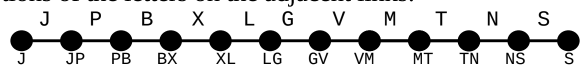
Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy. All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make "objects" at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label "the rest" can be used for some links, but this can be shown not to rescue the situation.

graph, including the sphere that the bloodhounds can see (smell?) into the "viewed" neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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 links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious "the rest" labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link's to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG's, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI'91 Proceedings,* 1991, 189-194.

To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

Requirement EVT1 (small views). The number of out-going links, or out-degree, of nodes in the viewing graph must be "small" compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

Requirement EVT2 (short paths). The distance (in number of links) between pairs of nodes 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. There are many "reasonable senses" one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

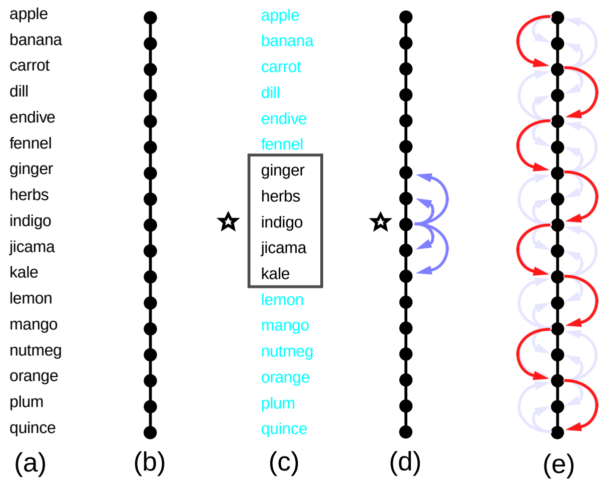


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),^{3}$$

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically "in the limit proportional to *1*", i.e., constant -- an excellent score. *O(n)* means "in the limit proportional to *n*"-- a pretty poor score. *O(log n)* would mean "in the limit proportional to *log n*"-- quite respectable.

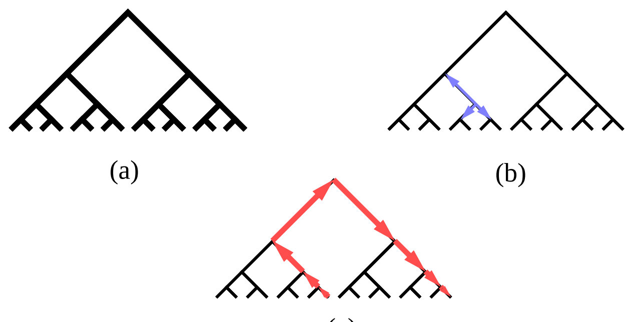


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

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

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4] One way to think about what has happened is

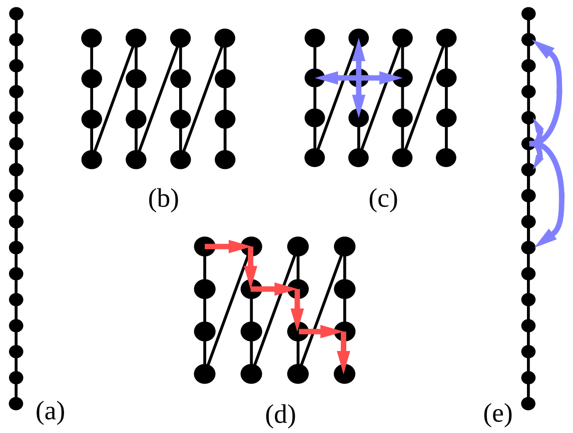


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

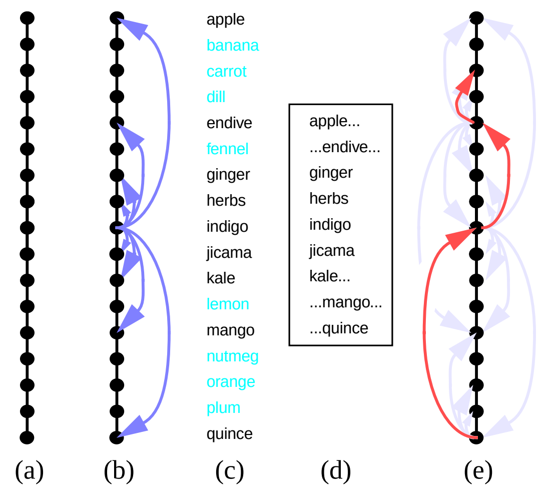


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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

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

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed,

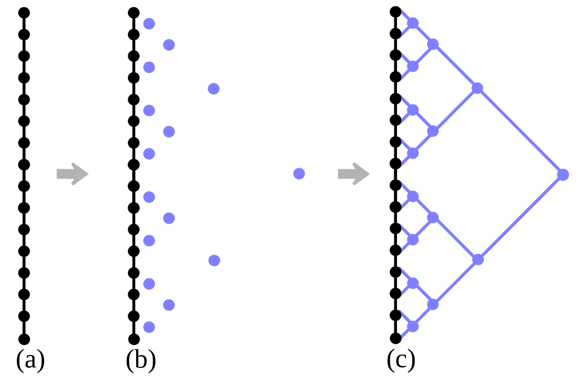


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

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

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n).* [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

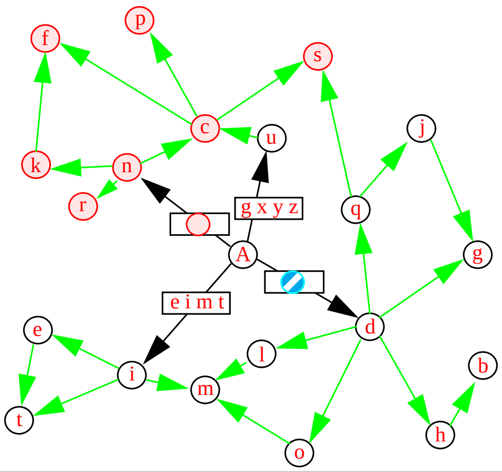
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently "leads to". If the navigator's target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	▨
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set.*)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it "leads to", as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This "enumeration" strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled "The Rest", which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

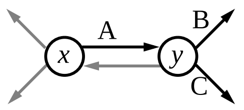


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to “BrownThings” and “LargeThings” but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy.

5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label “the rest” can be used for some links, but this can be shown not to rescue the situation.

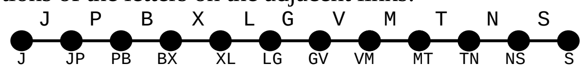
All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make “objects” at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

graph, including the sphere that the bloodhounds can see (smell?) into the "viewed" neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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* links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious "the rest" labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link's to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG's, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI'91 Proceedings*, 1991, 189-194.

To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

Requirement EVT1 (small views). The number of out-going links, or out-degree, of nodes in the viewing graph must be "small" compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

Requirement EVT2 (short paths). The distance (in number of links) between pairs of nodes 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. There are many "reasonable senses" one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

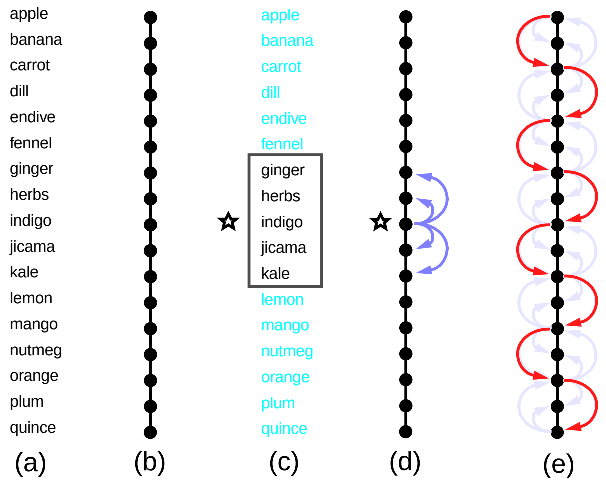


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),$$ [3]

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically "in the limit proportional to *1*", i.e., constant -- an excellent score. *O(n)* means "in the limit proportional to *n*"-- a pretty poor score. *O(log n)* would mean "in the limit proportional to *log n*"-- quite respectable.

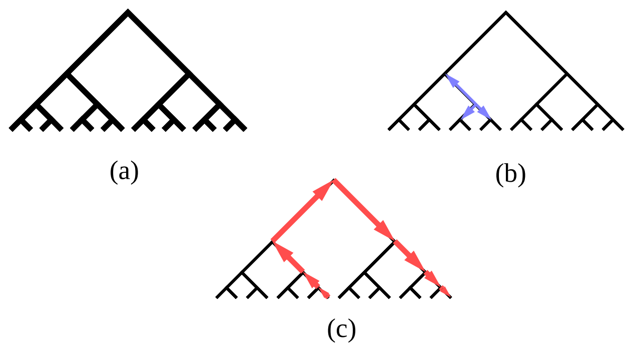


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

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

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4] One way to think about what has happened is

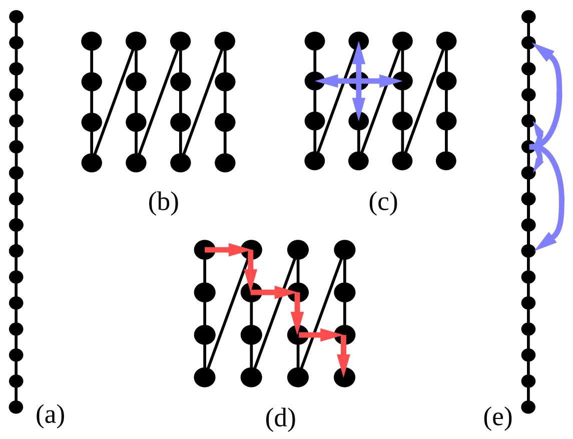


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

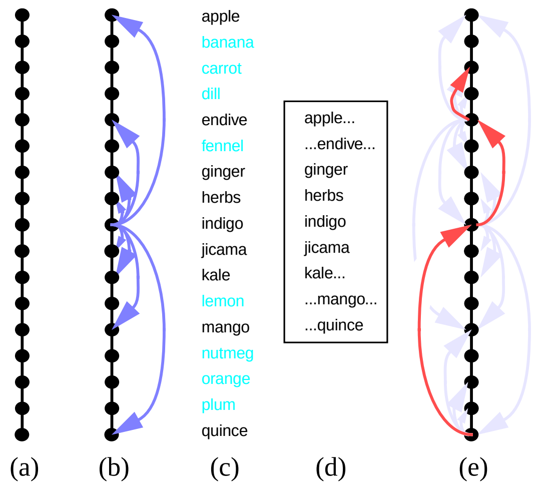


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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(FISHEYE-SAMPLED-LIST$_n$ *) = (O(log n), O(log n)).*

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed,

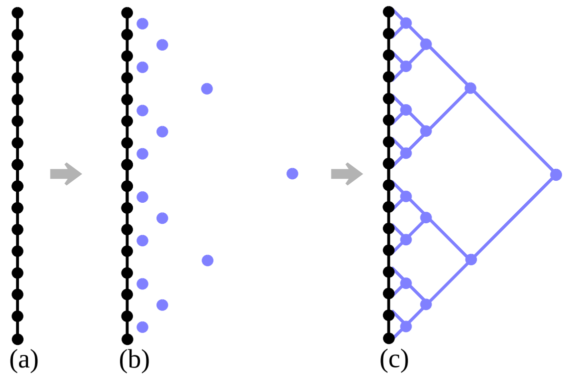


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

EVT(TREE-AUGMENTED-LIST$_n$ *) = (O(1), O(log n)).* ••

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n).* [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

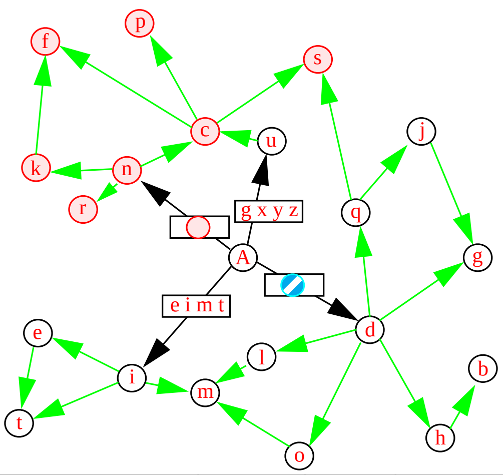
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently "leads to". If the navigator's target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	▨
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set.*)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it “leads to”, as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This “enumeration” strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled “The Rest”, which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

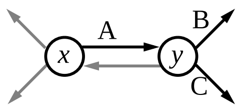


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to “BrownThings” and “LargeThings” but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

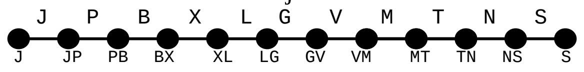
Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy. All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make “objects” at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label “the rest” can be used for some links, but this can be shown not to rescue the situation.

graph, including the sphere that the bloodhounds can see (smell?) into the "viewed" neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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* links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious "the rest" labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link's to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG's, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI'91 Proceedings,* 1991, 189-194.

To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

> *Requirement EVT1 (small views).* The number of out-going links, or out-degree, of nodes in the viewing graph must be "small" compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

> *Requirement EVT2 (short paths).* The distance (in number of links) between pairs of nodes 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. There are many "reasonable senses" one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

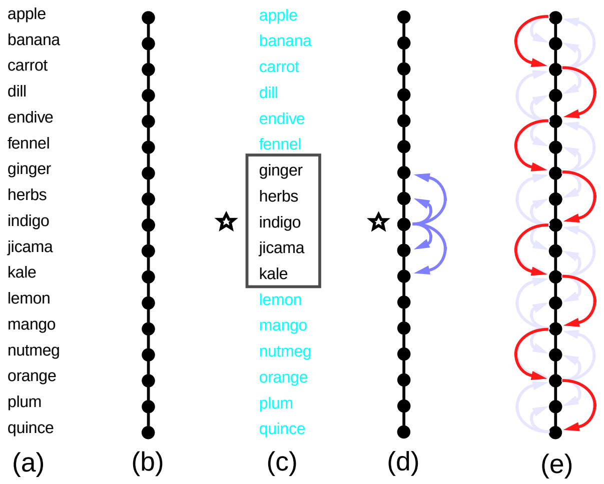


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),^3$$

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically "in the limit proportional to *1*", i.e., constant -- an excellent score. *O(n)* means "in the limit proportional to *n*"-- a pretty poor score. *O(log n)* would mean "in the limit proportional to *log n*"-- quite respectable.

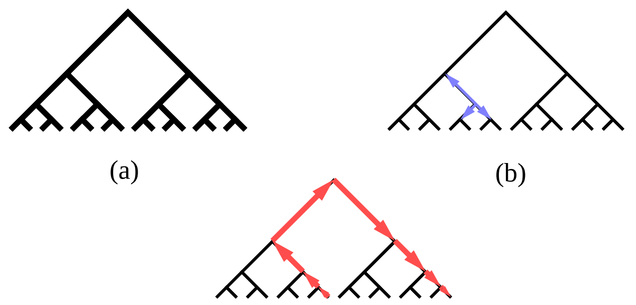


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

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

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4] One way to think about what has happened is

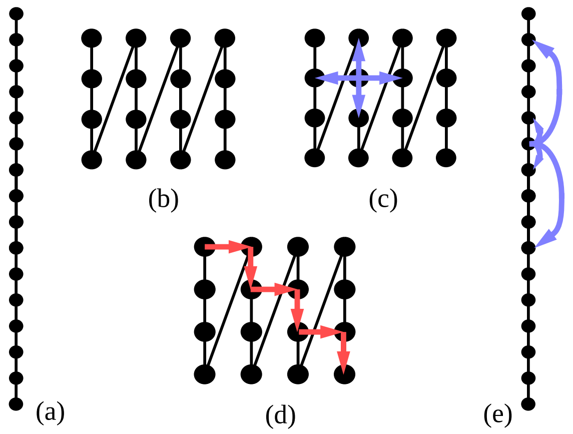


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

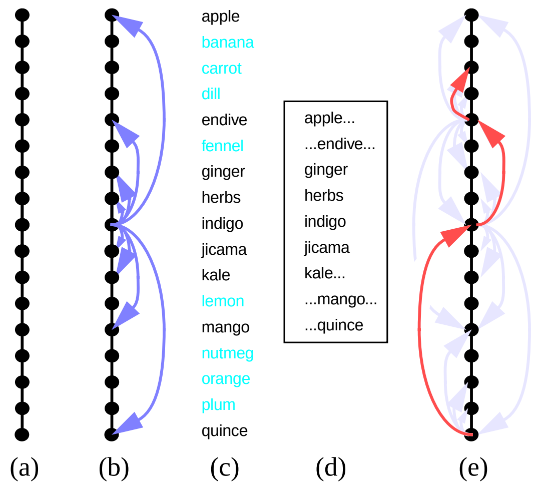


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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

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

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed,

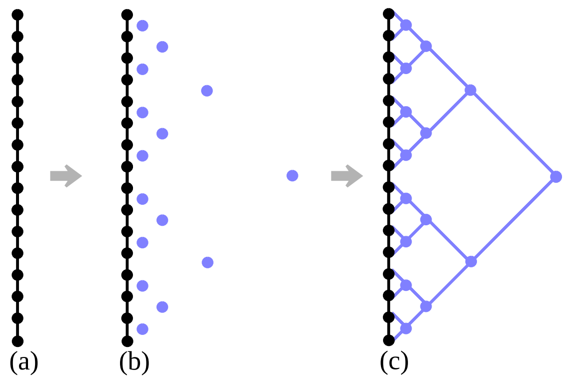


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

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

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n).* [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

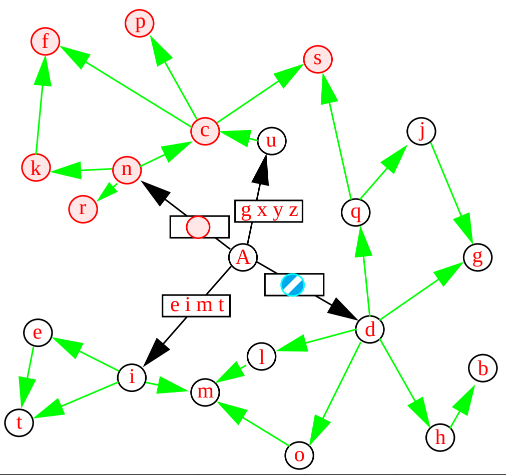
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently "leads to". If the navigator's target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	▨
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set*.)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it “leads to”, as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This “enumeration” strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled “The Rest”, which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

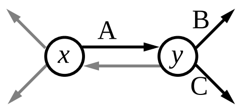


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to "BrownThings" and "LargeThings" but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

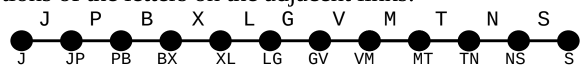
Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy. All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make "objects" at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label "the rest" can be used for some links, but this can be shown not to rescue the situation.

graph, including the sphere that the bloodhounds can see (smell?) into the "viewed" neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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* links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious "the rest" labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link's to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG's, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI'91 Proceedings*, 1991, 189-194.

To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

> *Requirement EVT1 (small views).* The number of out-going links, or out-degree, of nodes in the viewing graph must be "small" compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

> *Requirement EVT2 (short paths).* The distance (in number of links) between pairs of nodes 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. There are many "reasonable senses" one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

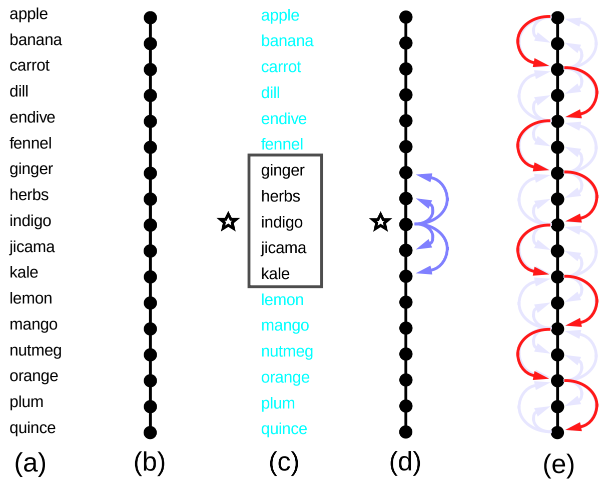


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),$$ [3]

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically "in the limit proportional to *1*", i.e., constant -- an excellent score. *O(n)* means "in the limit proportional to *n*"-- a pretty poor score. *O(log n)* would mean "in the limit proportional to *log n*"-- quite respectable.

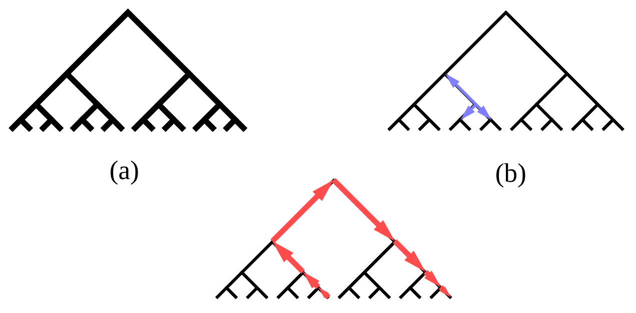


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

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

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4] One way to think about what has happened is

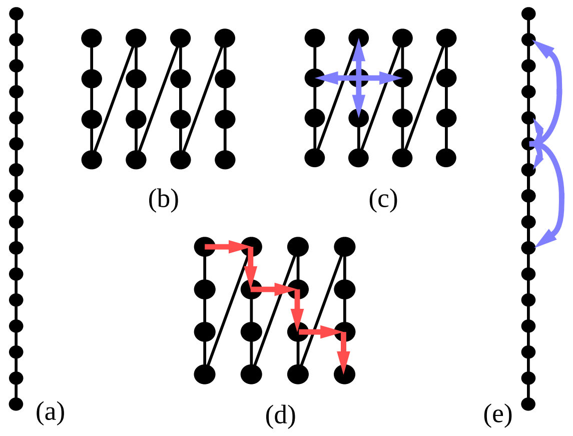


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

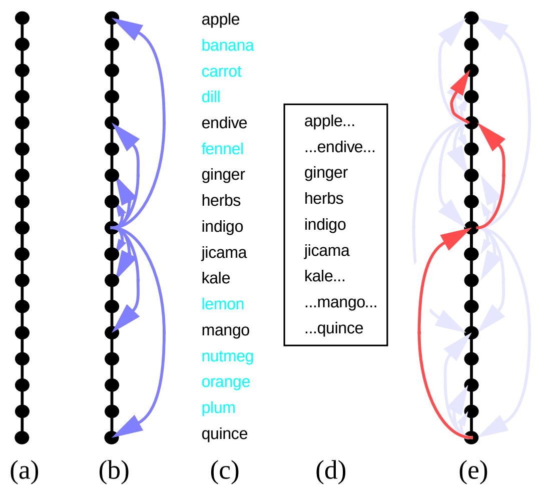


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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

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

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed, and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

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

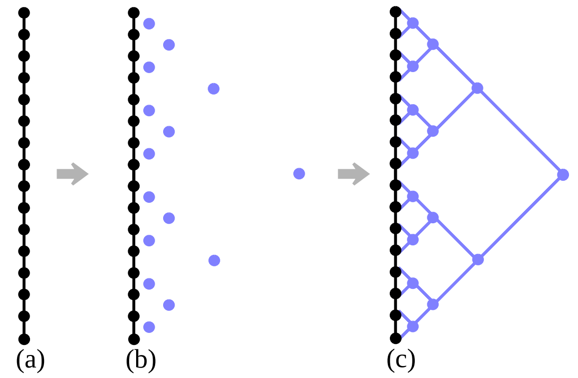


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n).* [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

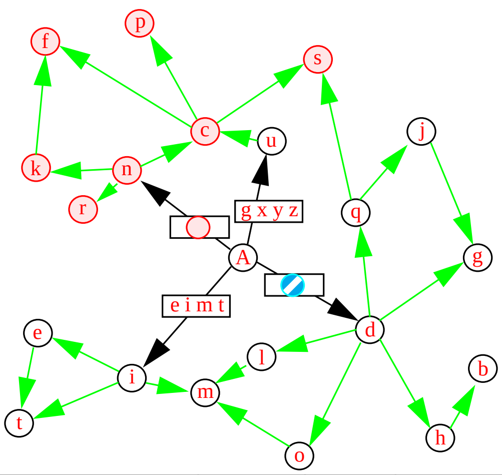
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently “leads to”. If the navigator’s target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	◍
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set*.)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it “leads to”, as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This “enumeration” strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled “The Rest”, which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

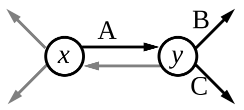


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to "BrownThings" and "LargeThings" but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy. All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

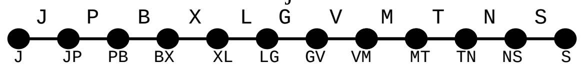
5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label "the rest" can be used for some links, but this can be shown not to rescue the situation.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make "objects" at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

graph, including the sphere that the bloodhounds can see (smell?) into the "viewed" neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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* links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious "the rest" labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link's to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG's, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI'91 Proceedings*, 1991, 189-194.

To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

> *Requirement EVT1 (small views).* The number of out-going links, or out-degree, of nodes in the viewing graph must be "small" compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

> *Requirement EVT2 (short paths).* The distance (in number of links) between pairs of nodes 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. There are many "reasonable senses" one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

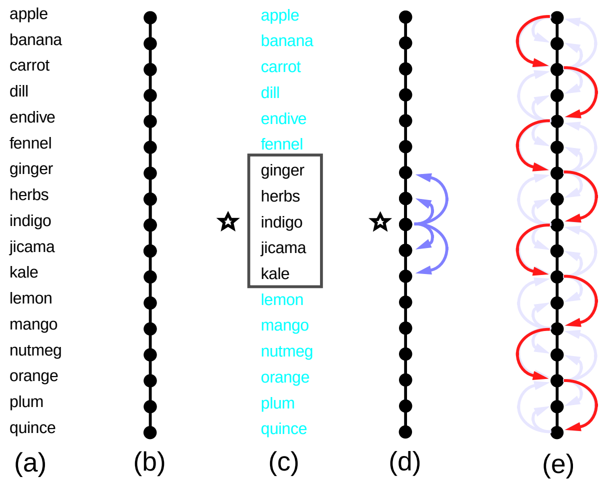


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),$$ [3]

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically "in the limit proportional to *1*", i.e., constant -- an excellent score. *O(n)* means "in the limit proportional to *n*"-- a pretty poor score. *O(log n)* would mean "in the limit proportional to *log n*"-- quite respectable.

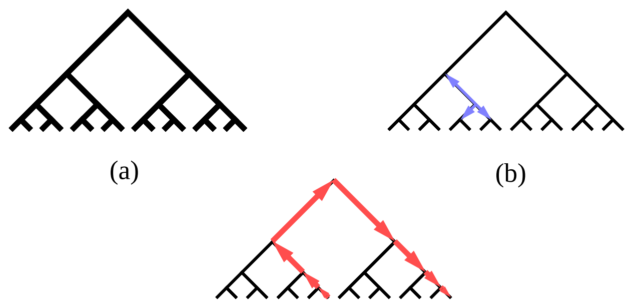


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

$$EVT(\text{ BALANCED-REGULAR-TREE}_n\ ) = (\ O(1), O(\log n)\ )$$ ••

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4]One way to think about what has happened is

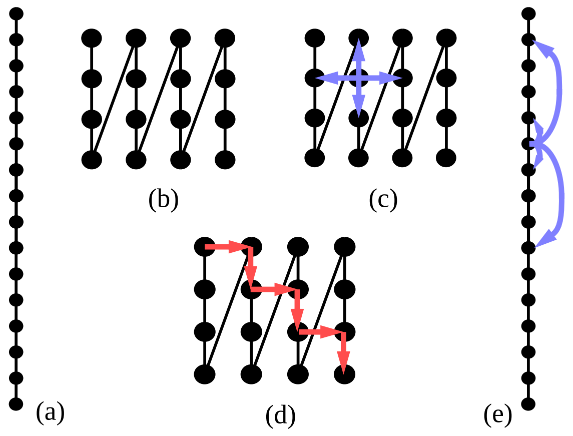


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

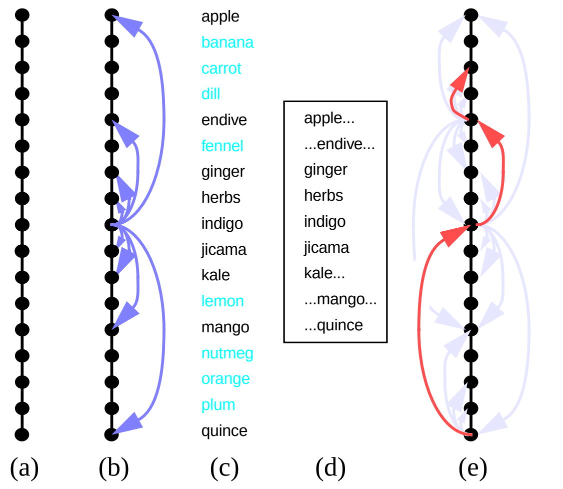


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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

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

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed,

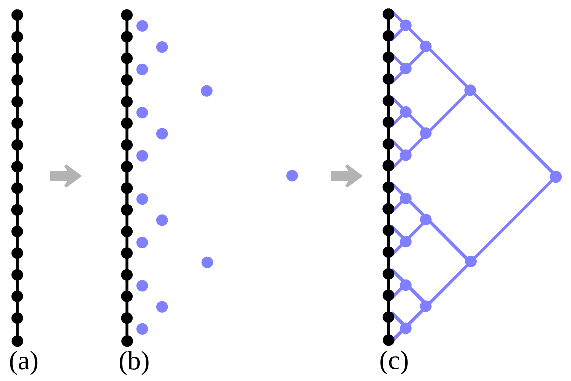


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

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

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n)*. [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

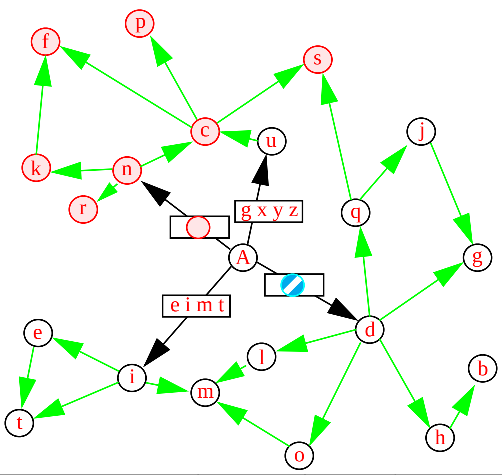
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently "leads to". If the navigator's target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	▨
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set.*)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it “leads to”, as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This “enumeration” strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled “The Rest”, which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

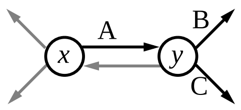


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to "BrownThings" and "LargeThings" but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy. All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

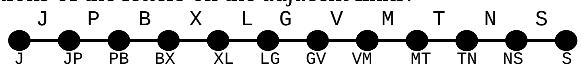
5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label "the rest" can be used for some links, but this can be shown not to rescue the situation.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make "objects" at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

graph, including the sphere that the bloodhounds can see (smell?) into the “viewed” neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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* links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious “the rest” labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link’s to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG’s, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI’91 Proceedings*, 1991, 189-194.

To appear in *Human Factors in Computing Systems, CHI'97 Conference Proceedings* (ACM), Atlanta GA, March 22-27, 1997.

Effective View Navigation

George W. Furnas
School of Information
University of Michigan
(313) 763-0076
furnas@umich.edu

ABSTRACT

In *view navigation* a user moves about an information structure by selecting something in the current view of the structure. This paper explores the implications of rudimentary requirements for effective view navigation, namely that, despite the vastness of an information structure, the views must be small, moving around must not take too many steps and the route to any target be must be discoverable. The analyses help rationalize existing practice, give insight into the difficulties, and suggest strategies for design.

KEYWORDS: Information navigation, Direct Walk, large information structures, hypertext, searching, browsing

INTRODUCTION

When the World Wide Web (WWW) first gained popularity, those who used it were impressed by the richness of the content accessible simply by wandering around and clicking things seen on the screen. Soon after, struck by the near impossibility of finding anything specific, global navigation was largely abandoned in place of search engines. What went wrong with pure navigation?

This work presented here seeks theoretical insight into, in part, the problems with pure navigational access on the web. More generally, it explores some basic issues in moving around and finding things in various information structures, be they webs, trees, tables, or even simple lists. The focus is particularly on issues that arise as such structures get very large, where interaction is seriously limited by the available resources of space (e.g., screen real estate) and time (e.g., number of interactions required to get somewhere): How do these limits in turn puts constraints on what kinds of information structures and display strategies lead to effective navigation? How have these constraints affected practice, and how might we live with them in future design?

We will be considering systems with static structure over which users must navigate to find things, e.g., lists, trees, planes, grids, graphs. The structure is assumed to contain elements of some sort (items in a list, nodes in a hypertext graph) organized in some logical structure. We assume that the interface given to the user is navigational, i.e., the user at any given time is "at" some node in the structure with a view specific to that node (e.g., of the local neighborhood), and has the ability to move next to anything in that current view. For example for a list the user might have window centered on a particular current item. A click on an item at the bottom of the window would cause that item to scroll up and become the new "current" item in the middle of the window. In a hypertext web, a user could follow one of the visible links in the current hypertext page.

In this paper we will first examine *view traversal*, the underlying iterative process of viewing, selecting something seen, and moving to it, to form a path through the structure.[1] Then we will look at the more complex *view navigation* where in addition the selections try to be informed and reasonable in the pursuit of a desired target. Thus view traversal ignores how to decide where to go next, for view navigation that is central.The goal throughout is to understand the implications of resource problems arising as structures get very large.

EFFICIENT VIEW TRAVERSIBILITY

What are the basic requirements for efficient view traversal? I.e., what are the minimal capabilities for moving around by viewing, selecting and moving which, if not met, will make large information structures, those encompassing thousands, even billions of items, impractical for navigation.

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 as dictated by the semantics of the domain.[2] For an ordered list this would just be line graph, with each item connected to the items which proceed and follow it. For a hyper-

1 This is the style of interaction was called a *direct walk* by Card, et al [2]. We choose the terminology "view traversal" here to make explicit the view aspect, since we wish to study the use of spatial resources needed for viewing.

2 More complete definitions, and proofs of most of the material in this paper can be found in [3]

text the logical structure graph would be some sort of web. We will assume the logical graph is finite.

We capture a basic property of the interface to the information structure in terms of the notion of a viewing *graph* (actually a directed graph) defined as follows. It has a node for each node in the logical structure. A directed link goes from a node *i* to node *j* if the view from *i* includes *j* (and hence it is possible to view-traverse from *i* to *j*). Note that the viewing graph might be identical to the logical graph, but need not be. For example, it is permissible to include in the current view, points that are far away in the logical structure (e.g., variously, the root, home page, top of the list).

The conceptual introduction of the viewing graph allows us to translate many discussion of views and viewing strategies into discussions of the nature of the viewing graph. Here, in particular, we are interested in classes of viewing-graphs that allow the efficient use of space and time resources during view traversal, even as the structures get very large: Users have a comparatively small amount of screen real estate and a finite amount of time for their interactions. For view traversal these limitations translate correspondingly into two requirements on the viewing graph. First, if we assume the structure to be huge, and the screen comparatively small, then the user can only view a small subset of the structure from her current location. In terms of the viewing graph this means, in some reasonable sense,

Requirement EVT1 (small views). The number of out-going links, or out-degree, of nodes in the viewing graph must be “small” compared to the size of the structure.

A second requirement reflects the interaction time limitation. Even though the structure is huge, we would like it to take only a reasonable number of steps to get from one place to another. Thus (again in some reasonable sense) we need,

Requirement EVT2 (short paths). The distance (in number of links) between pairs of nodes 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. There are many “reasonable senses” one might consider for formalizing these requirements. For analysis here we will use a worst case characterization. The Maximal Out-Degree (MOD, or largest out-degree of any node) will characterize how well EVT1 is met (smaller values are better), and the Diameter (DIA, or longest connecting path required anywhere) will characterize how well EVT2 is met (again, smaller is better). Summarized as an ordered pair,

$$EVT(G) = (MOD(G), DIA(G)),$$

we can use them to compare the traversability of various classes of view-traversable information structures.

•• *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 number of lines on either side. Thus a piece of the viewing graph, shown in (d), has links going from the starred node to all the nearby nodes that can be seen in the view as centered at that node. The complete viewing graph would have this replicated for all nodes in the logical graph.

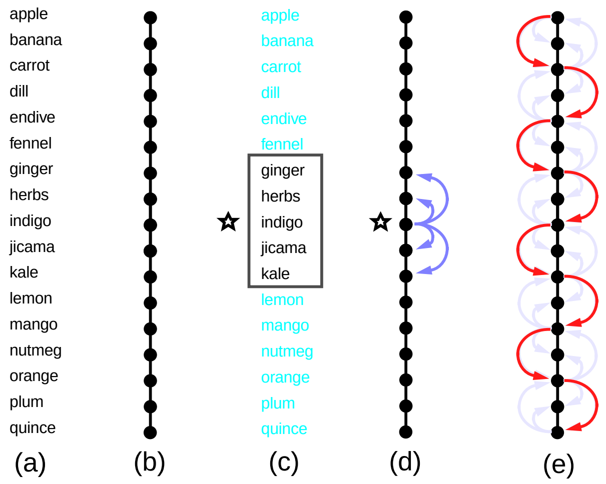


Figure 1. *(a) Schematic of an ordered list, (b) logical graph of the list, (c) local window view of the list, (d) associated part of viewing graph, showing that out degree is constant, (e) sequence of traversal steps showing the diameter of viewing graph is O(n).*

This viewing graph satisfies the first requirement, EVT1, nicely: Regardless of the length of the list, the view size, and hence the out-degree of the viewing graph, is always the small fixed size of the viewing window. The diameter requirement of EVT2, however, is problematic. Pure view traversal for a scrolling list happens by clicking on an item in the viewing window, e.g., the bottom line. That item 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 overall

$$EVT(\text{SCROLLING-LIST}_n) = (O(1), O(n)),$$ [3]

and, because of the diameter term, a scrolling list is not very Effectively View Traversable. This formalizes the intuition that while individual views in a scrolling list interface are reasonable, unaided scrolling is a poor interaction technique for even moderate sized lists of 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

Fortunately from a design standpoint, things need not be so bad. There are simple viewing structures that are highly EVT, and there are ways to fix structures that are not.

3 *O(1)* means basically “in the limit proportional to *1*”, i.e., constant -- an excellent score. *O(n)* means “in the limit proportional to *n*”-- a pretty poor score. *O(log n)* would mean “in the limit proportional to *log n*”-- quite respectable.

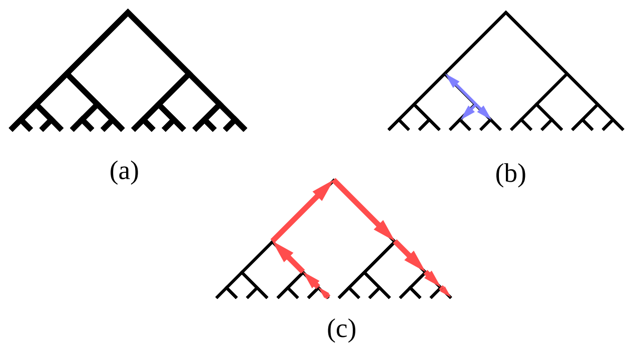


Figure 2. *An example of an Efficiently View Traversable Structure (a) logical graph of a balanced tree, (b) in gray, part of the viewing graph for giving local views of the tree showing the outdegree is constant, (c) a path showing the diameter to be O(log(n)).*

Some Efficiently View Traversable Structures

We begin by considering example structures that have good EVT performance, 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 (a balanced regular tree) appears in Figure 2(a). A typical tree viewing strategy makes visible from a given node 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 node in the viewing graph is a constant, one more than the branching factor, and we have nicely satisfied 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,

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

•• *example 3: local viewing of a hypercube*

Consider next a *k*-dimensional hypercube (not pictured) that might be used to represent the structure of a factorially designed set of objects, where there are *k* binary factors (cf. a simple but complete relational database with *k* binary attributes), e.g., a set of objects that are either big or small, red or green, round or square, in all various combinations. A navigational interface to such a data structure could give, from a node corresponding to one combination of attributes, views simply showing its neighbors in the hypercube, i.e., combinations differing in only one attribute. Whether this would be a good interface for other reasons or not, a simple analysis reveals that at least it would have very reasonable EVT behavior:

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

The conclusion of examples 2 and 3 is simply that, if one can coerce the domain into either a reasonably balanced and regular tree, or into a hypercube, view traversal will be quite efficient. Small views and short paths suffice even for very large structures. Knowing a large arsenal of highly EVT structures presumably will be increasingly useful as we have to deal with larger and larger information worlds.

Fixing Non-EVT Structures

What can one do with an information structure whose logical structure does not directly translate into a viewing graph that is EVT? The value of separating out the viewing graph from the logical graph is that while the domain semantics may dictate the logical graph, the interface designer can often craft the viewing graph. Thus we next consider a number of strategies for improving EVT behavior by the design of good viewing graphs. We illustrate by showing several ways to improve the view navigation of lists, then mentioning some general results and observations.

•• *example 4: fixing the list (version 1) - more dimensions*

One strategy for improving the View Traversability of a list is to fold it up into two dimensions, making a multi-column list (Figure 3).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, 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

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

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 or a page of the phone book). The advantage is presumably that the eye (or viewing window) has to move a much shorter distance to get from one part to another.[4] One way to think about what has happened is

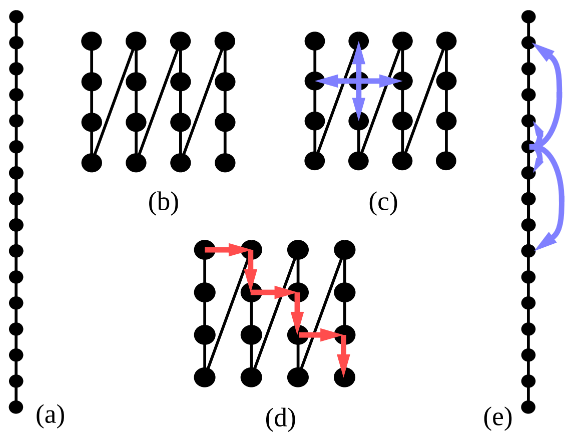


Figure 3. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) the list is folded up in 2-D (c) part of the viewing graph showing the 2-D view-neighbors of Node6 in the list: out degree is O(1), (d) diameter of viewing graph is now reduced to O(sqrt(n)). (e) Unfolding the list, some view-neighbors of Node6 are far away, causing a decrease in diameter.*

4 Some really long lists, for example a metropolitan 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 how far one would have to move from the beginning to the end of the list if one only had 1D or 2D in which to place all those numbers! A similar analysis explains how Cone Trees [6] provide better traversability using 3-D.

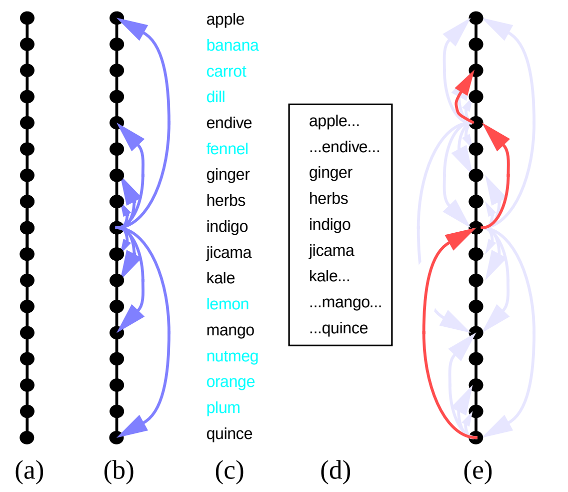


Figure 4. *Fixing the list viewer. (a) logical graph of the ordered list again, (b) part of viewing graph of fish-eye sampled list, showing that out degree is O(log(n)), (c) sample actually selected from list (d) view actually given, of size O(log(n)), (e) illustration of how diameter of viewing graph is now O(log(n)).*

illustrated in Figure 3 (e), where the part of the viewing graph in (c) is shown in the unfolded version of the list.••

The critical thing to note in the example is that some of the links of the viewing graph point to nodes 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 sampling*

It is possible to use non-local viewing links to improve even further the view-traversability of a list. Figure 3 shows a viewing strategy where the nodes included in a view are spaced in a geometric series. That is, from the current node, one can see the nodes that are at a distance 1, 2, 4, 8, 16,... away in the list. 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

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

Note that variations of this fisheye sampling strategy can yield good EVT performance for many other structures, including 2D and 3D grids. ••

The lesson from examples 4 and 5 is that even if the logical structure is not EVT, it is possible to make the viewing structure EVT by adding long-distance links.

•• *example 6: fixing the list (version 3) - tree augmentation*

In addition to just adding links, one can also adding nodes 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 links needed,

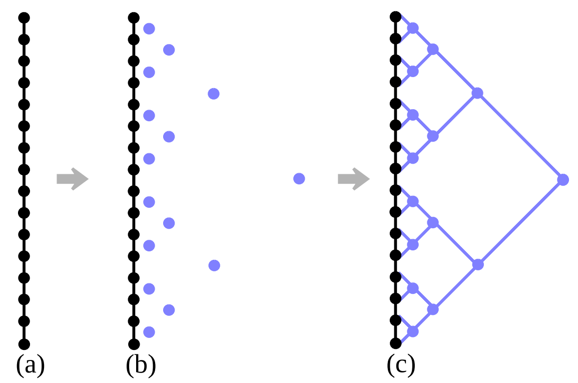


Figure 5. *Improving EVT of a list by adding a tree. The resulting structure has constant viewsize but logarithmic diameter*

and hence the general outdegree. For example, one can glue a structure known to be EVT onto a given non-EVT structure. In Figure 2 a tree is glued onto the side of the list and traversal is predominantly through the tree. Thus in Figure 2, new nodes are introduced (*O(n)*), and viewing links are introduced in the form of a tree. Since the outdegrees everywhere are of size ≤3, and logarithmic length paths now exist between the original nodes by going through the tree, we get

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

Although there is not enough space here for details, we note in passing that an EVT analysis of zoomable interfaces is valuable in clarifying one of their dramatic advantages -- the diameter of the space is reduced from *O(sqrt(n))* to *O(log n).* [3][4]

Remarks about Efficient View Traversability

Efficient View Traversability is a minimal essential condition for view navigation of very large information structures. In a straight forward way it helps to explain why simple list viewers do not scale, why phone books and cone-trees exploit 3D, why trees and fisheyes and zooms all help.

EVT analysis also suggests strategies for design. One can try to coerce an information world into a representation which naturally supports EVT1 and EVT2, e.g., the common practice of trying to represent things in trees. Alternatively one can fix a poor structure by adding long-distance links, or adding on another complete structure. Note that in general, the impact of selectively adding links can be much greater on decreasing diameter than on increasing view sizes, to net positive effect. One result of this simple insight is a general version of the tree augmentation strategy. In general terms, gluing any good EVT graph onto a bad one will make a new structure as good as the good graph in diameter, and not much worse than the worse of the two in outdegree. I.e., always remember the strategy of putting a traversable infrastructure on an otherwise unruly information structure!

Efficiently view traversable structures have an additional interesting property, *"jump and show":* an arbitrary non-navigational jump (e.g., as the result of a query search) from one location to another has a corresponding view traversal version with a small rendering: a small number of steps each requir-

ing a small view will suffice. Thus a short movie or small map will show the user how they could have done a direct walk from their old location to their new one. A similar concept was explored for continuous Pan&Zoom in [4].

NAVIGABILITY

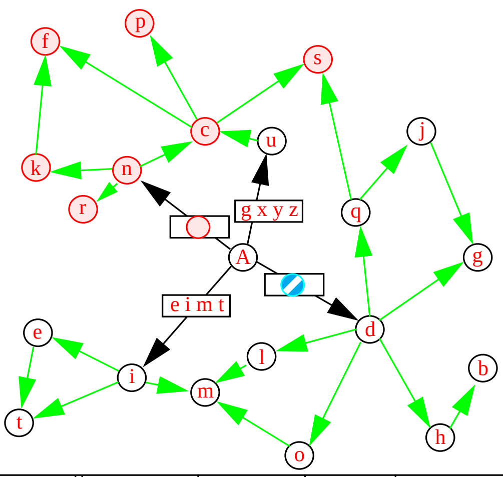
Efficient view traversability is not enough: it does little good if a short traversal path to a destination exists but that path is unfindable. It must be possible somehow to read the structure to find good paths; the structure must be *view navigable.*

For analysis we imagine the simple case of some *navigator* process searching for a particular target, proceeding by comparing it to information associated with each outlink in its current view (*outlink-info*, e.g. a label). From this comparison, it decides which link to follow next.

In this paper we will explore an idealization, *strong navigability* , requiring that the structure and its outlink-info allow the navigator (1) to find the shortest path to the target (2) without error and (3) in a history-less fashion (meaning it can make the right choice at each step by looking only at what is visible in the current node). We examine this case because it is tractable, because it leads to suggestive results, and because, as a desirable fantasy, its limits are worth understanding.

To understand when strong view navigation is possible, a few definitions are needed. They are illustrated in Figure 6 .

Consider a navigator at some node seeking the shortest path to a target. A given link is a defensible next step only for certain targets -- the link must be on a shortest path to those targets. We call this set of targets the *to-set* of the link, basically the targets the link efficiently “leads to”. If the navigator’s target is in the to-set of a link, it can follow that link.



Link	A-->n	A-->u	A-->i	A-->d
to-set	{c,f,k,p,r,s}	{c,f,u,p,s}	{e,i,m,t}	{b,d,g,h,j,l,o,q,s}
outlink-info	●	g x y z	e i m t	▨
inferred to-set	<c,f,k,p,r,s>	<g,x,y,z>	<e,i,m,t>	<?>

Figure 6 *The outlink-info for link A-->i is an enumeration, and for A-->n is a feature (a shaded circle). These are both well matched. The link info for links to the right of A is not-well matched. The residue of f at A is the shaded-circle label. The residue of e at A is its appearance in the enumeration label. The node g has residue in the upper right enumeration label at A, but it is not good residue. The node h has no residue at A.*

We assume, however, that the navigator does not know the to-set of a link directly; it is a global property of the structure of the graph. The navigator only has access to the locally available outlink-info which it will match against its target to decide what link to take. We define the *inferred-to-set* of a link to be the set of all target nodes that the associated outlink-info would seem to indicate is down that link (the targets that would match the outlink-info), which could be a different set entirely.

In fact, we say that the outlink-info of a link is *not misleading with respect to a target* when the target being in the *inferred-to-set* implies it is in the true *to-set*, or in other words when the outlink-info does not mislead the navigator to take a step it should not take. (Note that the converse is not being required here; the outlink-info need not be complete, and may underspecify its *true-to-set.*)

Next we say that the outlink-info of node as a whole is said to be *well-matched with respect to a target* if none of its outlink-info is misleading with respect to that target, and if the target is in the inferred-to-set of at least one outlink. Further we say that the outlink information at a node is simply *well-matched* iff it is well-matched with respect to all possible targets.

We now state the following straightforward proposition:

> *Proposition (navigability):* The navigator is guaranteed to always find shortest paths to targets iff the outlink-info is everywhere well-matched.

Hence, the following requirement for a strongly navigable world:

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

The critical observation in all this for designing navigable 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, including the WWW: Their link-labels indicate adjacent nodes and do not tell what else lies beyond, in the whole to-set. 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. As we will see shortly, this is a difficult requirement in practice.

First, however, we turn the analysis on its head. The perspective 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, with the assumption that targets want a world in which they can be found. This complementary perspective brings up the important notion of *residue* or *scent* .The resi-

due or scent of a target is the remote indication of that target in outlink-info throughout the information structure. More precisely, a target has residue at a link if the associated outlink-info would lead the navigator to take that link in pursuit of the given target, i.e., to put the target in the inferred-to-set of the link. If the navigator was not being mislead, i.e, the outlink-info was well-matched for that target, then we say the residue was *good residue* for the target.(Refer back to the caption of Figure 6).

An alternate formulation of the Navigability proposition says that in order to be findable by navigation from anywhere in the structure, a target must have good residue at every node. I.e., in order to be able to find a target, the navigator must have some scent, some residue, of it, no matter where the navigator is, to begin chasing down its trail.

Furthermore, if every target is to be findable, we need the following requirement, really an alternate statement of 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 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[5]. If you zoom out too far, your target can become unrecognizable, or disappear entirely leaving no residue, and you cannot navigate back to it. This has lead to a notion of *semantic zooming [1] [5]* , where the appearance of an object changes as its size changes so that it stays visually comprehensible, or at least visible -- essentially a design move to preserve good residue.

The VN1 requirements are difficult 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 global residue distributions would be to have an exhaustive list, or enumeration, of the to-set as the outlink-info for each link (i.e., each link is labeled by listing the complete set of things it “leads to”, as in the label of the lower left outlink from node *A* in). Thus collectively between all the outlinks at each node there is a full listing of the structure, and such a complete union list must exist at each node. This “enumeration” strategy is presumably not feasible for view navigation since, being $O(n^2)$, it does not scale well. Thus, the outlink-info must somehow accurately specify the corresponding to-set in some way more efficient than enumeration, using more conceptually complex representations, requiring semantic notions like attributes (Red) and abstraction (LivingThings).

The issues underlying good residue, its representation and distribution, are intriguing and complex. Only a few modest observations will be listed here.

Remarks on View Navigability

Trees revisited. One of the most familiar navigable information structures is a rooted tree in the form of classification hierarchies like biological taxonomies or simple library classification schemes like the dewey decimal system. In the traversability section of this paper, balanced trees in their completely unlabeled form were hailed as having good traversal properties just as graphs. Here there is an entirely different point: a systematic labeling of a rooted tree as a hierarchy can make it in addition a reasonably navigable structure. Starting at the root of a biological taxonomy, one can take Cat as a target and decide in turn, is it an Animal or a Plant, is it a Vertebrate or Invertebrate, etc. and with enough knowledge enough about cats, have a reasonable (though not certain!) chance of navigating the way down to the Cat leaf node in the structure. This is so familiar it seems trivial, but it is not.

First let us understand why the hierarchy works in terms of the vocabulary of this paper. There is well matched out-link info at each node along the way: the Vertebrate label leads to the to-set of vertebrates, etc. and the navigator is not misled. Alternately, note that the Cat leaf-node has good residue everywhere. This is most explicit in the Animal, Vertebrate, Mammal,... nodes along the way from the root, but there is also implicit good residue throughout the structure. At the Maple node, in addition to the SugarMaple and NorwayMaple children, neither of which match Cat, there is the upward outlink returning towards the root, implicitly labeled “The Rest”, which Cat does match, and which is good residue. This superficial explanation has beneath it a number of critical subtleties, general properties of the semantic labeling scheme that rely on the richness of the notion of cat, the use of large semantic categories, and the subtle web woven from of these categories. These subtleties, all implied by the theory of view navigation and efficient traversability not only help explain why hierarchies work when they do, but also give hints how other structures, like hypertext graphs of the world wide web, might be made navigable.

To understand the navigation challenge a bit, consider the how bad things could be.

The spectre of essential non-navigability. Consider that Requirement VN2 implies that typically the minimum description length of the to-sets must be small compared to the size of those sets. In information theory that is equivalent to requiring that the to-sets are not random. Thus,

•• *example 7: non-navigable 1 - completely unrelated items.*

A set of *n* completely unrelated things is intrinsically not navigable. To see this consider an abstract alphabet of size *n*. Any subset (e.g., a to-set) is information full, with no structure or redundancy, and an individual set cannot be specified except by enumeration. As a result it is not hard to show there is no structure for organizing these *n* things, whose outlinks can be labeled except with predominant

use of enumeration[5], and so overall VN2 would be violated. ••

Such examples help in understanding navigability in the abstract, and raise the point that insofar as the real world is like such sets (is the web too random?), designing navigable systems is going to be hard. Having set two extremes, the reasonably navigable and the unnavigable, consider next a number of general deductions about view navigation.

Navigability requires representation of many sets. Every link has an associated toset that must be labeled. This means that the semantics of the domain must be quite rich to allow all these sets to have appropriate characterizations (like, RedThings, Cars, ThingsThatSleep). Similarly since a target must have residue at every node, each target must belong to *n* of these sets -- in stark contrast to the impoverished semantics of the purely random non-navigable example.

Navigability requires an interlocking web of set representations. Furthermore, these to-sets are richly interdependent. Consider the local part of some structure shown in Figure 7. Basically, the navigator should go to *y* to get to the

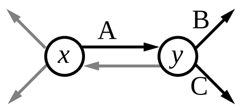


Figure 7. *Two adjacent nodes, x and y, in a structure. The to-sets associated with each outlink are labeled in upper case.*

targets available from *y*. In other words the to-set, A, out of *x* , is largely made up of the to-sets *B* and *C* out of neighboring *y*. (The exceptions, which are few in many structures, are those targets with essentially an equally good path from *x* and *y*.) This indicates that a highly constrained interlocking web of tosets and corresponding semantics and labels must be woven. In a hierarchy 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. Animals leads, not to "BrownThings" and "LargeThings" but to Vertebrates and Invertebrates -- a conceptual partition the navigator can decode mirroring an actual partition in the toset. Other structures do not often admit such nice partitioning semantics. It is unclear what other structures have to-sets and webs of semantic labelings that can be made to mirror each other.

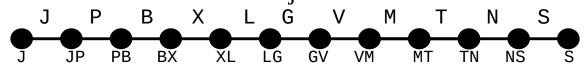
Residue as a shared resource. Since ubiquitous enumeration is not feasible, each target does not get its own explicit listing in outlink-info everywhere. It follows that in some sense, targets must typically share residue. The few bits of outlink-info are a scarce resource whose global distribution must be carefully structured, and not left to grow willy-nilly. To see this consider putting a new page up on the web. In theory, for strong navigability, every other node in the net must suddenly have good residue for this new page! Note how cleverly this can be handled in a carefully crafted hierarchy. All the many vertebrates share the short Vertebrate label as residue. Global distribution is maintained by the careful placement of new items: put a new vertebrate in the right branch of the tree, and it shares the existing good residue. It is probably no accident that the emerging large navigable substructures over the web, e.g. Yahoo!, arise in a carefully crafted hierarchical overlay with careful administrative supervision attending to the global distribution of this scare residue resource.

Similarity-based navigation. One interesting class of navigable 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 simply indicates the object at other end of link. The navigator can compute the similarity between objects, and chooses the outlink whose associated object is most similar to its ultimate target, in this way hill-climbing up a similarity gradient until the target is reached. One might navigate through color space in this way, moving always towards the neighboring color that is most similar to the target. Or one might try to navigate through the WWW by choosing an adjacent page that seems most like what one is pursuing (this would be likely to fail in general).

Similarity based navigation requires that nodes for similar objects be pretty closely linked in the structure, but that is not sufficient. There can be sets of things which have differential similarity (not completely unrelated as in the non-navigable example 6), and which can be linked to reflect that similarity perfectly, but which are still fundamentally non-navigable, essentially because all similarity is purely local, and so there is no good-residue of things far away.

•• *example 8: non-navigable set 2 - locally related structure.*

This example concerns sets with arbitrary similarity structure but only local semantics, and that are hence non-navigable.Take any graph of interest, for example the line graph below. Take an alphabet as large as the number of links in the graph, and randomly assign the letters to the links. Now make "objects" at the nodes that are collections of the letters on the adjacent links:



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 good-residue of things far away ••

This example might be a fair approximation to the WWW -- pages might indeed be similar, or at least closely related, to their neighbors, yet it is in general a matter of relatively small, purely local features, and cannot support navigation.

Weaker models of navigability. Suppose we were to relax strong navigability, for example abandoning the need for every target to have residue at the current node. Even then resource constraints dictate that it be possible to explore in a small amount of time and find appropriate good residue.This suggests that this relaxation will not dramatically alter the general conclusions. Imagine that you could sit at a node and send out a pack of seeing-eye bloodhounds looking for scent at each step. This really amounts to just changing the viewing

5 Technically, use of enumeration must dominate but need not be ubiquitous. Some equivalent of the short label "the rest" can be used for some links, but this can be shown not to rescue the situation.

graph, including the sphere that the bloodhounds can see (smell?) into the "viewed" neighbors. The constraints on how many hounds and how long they can explore basically remains a constraint on outdegree in the revised graph.

Combining EVT + VN = Effective View Navigability (EVN)

If we want an information structure that is both efficiently traversable, and is strongly view navigable, then both the mechanical constraints 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.

Large scale semantics dominate. 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* links 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.

The significance of this is that VN1 requires that outlink-info faithfully represent these large to-sets. If we assume the representations are related to the semantics of objects, and that representations of large sets are in some sense high level semantics, it follows that high level semantics play a dominant role in navigable structures. In a hierarchy this is seen in both the broad category labels like Animal and Plant, and in the curious "the rest" labels. The latter can be used in any structure, but are quite constrained (e.g., they can only be used for one outlink per node), so there is considerable stress on more meaningful coarse-grain semantics. So if for example the natural semantics of a domain mostly allow the specification of small sets, one might imagine intrinsic trouble for navigation. (Note that Example 8 has this problem.)

Carving up the world efficiently. 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. Enforcing the diameter requirement of EVT2 means the neighboring to-sets have to differ more dramatically. Consider by contrast the to-sets of the line graph, a graph with bad diameter. There the to-sets change very slowly as one moves along, with only one item being eliminated at a time. It is possible to show (see [3]) that under EVT2 the overlap pattern of to-sets must be able to whittle down the space of alternatives in a small number of intersections. The efficiency of binary partitioning is what makes a balanced binary tree satisfy EVT2, but a very unbalanced one not. Correspondingly an efficiently view navigable hierarchy has semantics that partition into equal size sets, yielding navigation by fast binary search. More generally, whatever structure, the semantics of the domain must carve it up efficiently.

Summary and Discussion

The goal of the work presented here has been to gain understanding of view navigation, with the basic premise that scale issues are critical. The simple mechanics of traversal require design of the logical structure and its viewing strategy so as to make efficient uses of time and space, by coercing things into known EVT structures, adding long distance links, or gluing on navigational backbones. Navigation proper requires that all possible targets have good residue throughout the structure. Equivalently, labeling must reflect a link's to-set, not just the neighboring node. This requires the rich semantic representation of a web of interlocking sets, many of them large, that efficiently carve up the contents of the space.

Together these considerations help to understand reasons why some information navigation schemes are,

bad: the web in general (bad residue, diameter), simple scrolling lists (bad diameter)

mixed: geometric zoom (good diameter, poor residue),

good: semantic zoom (better residue), 3D(shorter paths), fisheyes (even shorter paths), balanced rooted trees (short paths and possible simple semantics)

The problem of global residue distribution is very difficult. The taxonomies implemented in rooted trees are about the best we have so far, but even they are hard to design and use for all purposes. New structures should be explored (e.g., hypercubes, DAG's, multitrees), but one might also consider hybrid strategies to overcome the limits of pure navigation, including synergies between query and navigation. For example, global navigability may not be achievable, but local navigability may be - e.g., structures where residue is distributed reasonably only within a limited radius of a target. Then if there is some other way to get to the right neighborhood (e.g., as the result of an query), it may be possible to navigate the rest of the way. The result is query initiated browsing, an emerging paradigm on the web. Alternatively, one might ease the residue problem by allowing dynamic outlink-info, for example relabeling outlinks by the result of an ad-hoc query of the structure.

Acknowledgments. This work was supported in part by ARPA grant N66001-94-C-6039.

REFERENCES

1. Bederson, B. B. and Hollan, J. D., PAD^{++}: 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. 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.
3. Furnas, G.W., Effectively View-Navigable Structures. Paper presented at the 1995 Human Computer Interaction Consortium Workshop (HCIC95), Snow Mountain Ranch, Colorado Feb 17, 1995. Manuscript available at `http://http2.si.umich.edu/~furnas/POSTSCRIPTS/EVN.HCIC95.workshop.paper.ps`
4. 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.
5. 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.
6. Robertson, G. G., Mackinlay, J.D., and Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. *CHI'91 Proceedings*, 1991, 189-194.