Update (June 2004): many of the ideas in the below document have been revised and/or extended in a paper that has been accepted for publication at the Hypertext and Hypermedia 2004 conference. For a copy of that paper, which includes an updated version of Figure 18 below, see http://www.dgp.utoronto.ca/~mjmcguff/research/

A Graph-Theoretic Introduction to Ted Nelson's Zzstructures

Michael J. McGuffin
January 2004

---

1. Introduction

Zzstructure is the generic name for Ted Nelson's ZigZag(tm) [13,6,23,10], an unusual way of linking and organizing information. Zzstructures are one aspect of Nelson's vision of what hypertext should be (see, for example, Xanadu [11,12,5]). Many uses for zzstructures have been proposed, from organizing personal schedules [14], to visualizing genealogical family "trees" [26], to "cell programming" [22] (an example of a cellular programming language is Clang [25]).

As a data structure, Zzstructures allow nodes (or *cells*) of information to be linked into lists, cycles, grids (i.e. spreadsheet-like tables), trees, or all of these things simultaneously. There are also interesting visualization techniques that have been proposed for zzstructures.

This document describes the basic structure of zzstructures and a simple way of visualizing them. It is partly a compilation of information found in other documents, and partly a rephrasing of previous descriptions of zzstructures. Unlike other introductions [15,27] to zzstructures, this document describes zzstructures as a kind of graph (specifically, a directed multigraph with coloured edges) and compares them to other kinds of graphs (multitrees [3] and polyarchies [7,8,9]). The hope is that by explaining concepts with standard graph theory terminology, this document might enable zzstructures to be more widely and better understood by computer scientists.

Apart from trying to make zzstructure concepts more accessible and easier to understand, this document also contains some minor original contributions, which may have been anticipated by Nelson or others. First, the display of "incidental edges" and elisions in Figure 8. (These, or similar or better features, may already be supported by Gzz [24] -- a software package which I have not yet played with. Also, slide 16 of [23] alludes to indicating when cells are connected or not connected to other cells which may or may not be visible.) Second, I propose H^+- and I^+-views in Figure 9. Third, the classification in Figure 18 is, to my knowledge, the first taxonomy to compare zzstructures with multitrees, polyarchies, and other structures. Fourth, the last section of this document identifies some open problems and areas for future work in zzstructures.

---

2. Definition and Interpretation of Zzstructures

2.1. A graph-oriented definition

A brief review of some standard graph terminology (see Cormen et al. [1] for an excellent

introduction):

> A graph is an ordered pair G=(V,E) of a set V of nodes (or vertices) and a set E of edges (or arcs). Each edge is a pair of nodes. Graphically, the nodes of a graph may be embedded in a 2D space, and the edges between nodes may be represented as straight or curved line segments connecting nodes.
>
> A directed graph, or digraph, is a graph where the edges are *ordered* pairs of nodes, i.e. E is a subset of VxV. Hence, each edge can be thought of as a one-way connection from one node to another. Graphically, this is represented by drawing arrow heads on edges.
>
> A multigraph is a graph where E is a multiset, i.e. there are potentially multiple instances of edges between identical pairs of nodes.

Zzstructures are a kind of directed multigraph, where the edges are coloured or typed (i.e. each edge has a colour or type associated with it), and are subject to the following restriction:

> Restriction R: each node in a zzstructure may have at most one incoming edge of each colour, and at most one outgoing edge of each colour.

Thus, if red and blue are edge colours, then each node in a zzstructure may have 0 or 1 incoming red edges, 0 or 1 incoming blue edges, 0 or 1 outgoing red edges, and 0 or 1 outgoing blue edges.

Because of restriction R, the edges of common colours in a zzstructure form non-branching, non-intersecting paths and/or cycles within the graph. Because each node has at most one incoming and one outgoing edge of each colour, there is a unique way to move backward or forward along the path of each colour through that node.

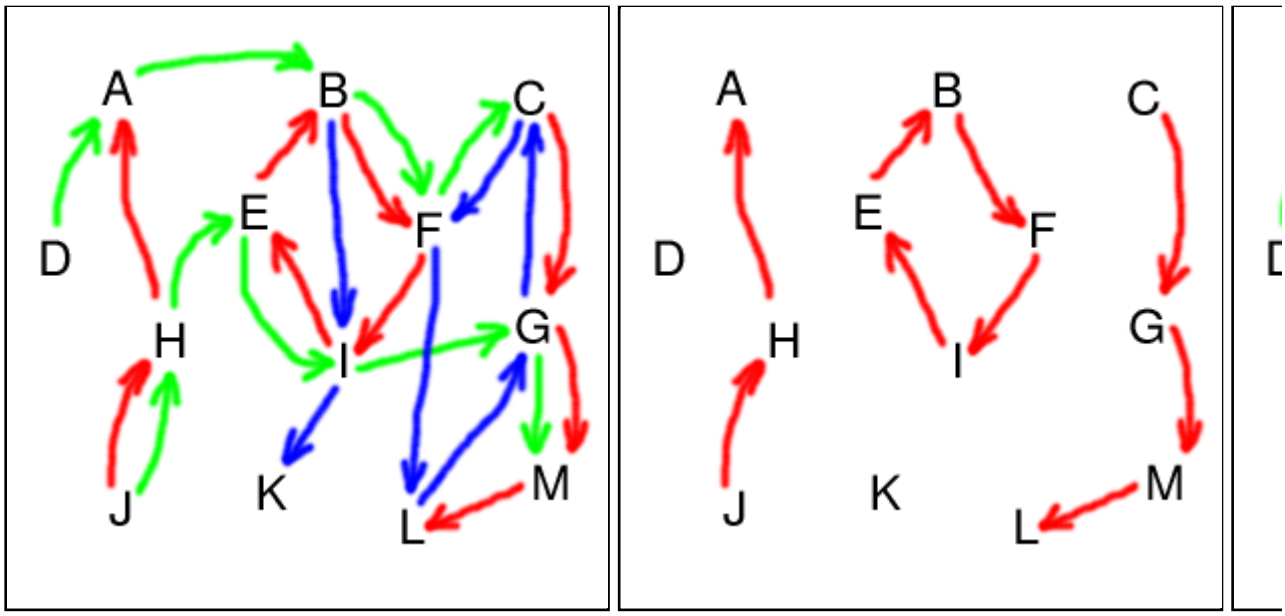

Figure 1. Above at left is an example zzstructure involving 3 edge colours. The other 3 images show the edge subsets of each colour. Notice how each subset of edges is a set of non-intersecting paths and/or cycles.

Note that, in the present depictions of zzstructures, one-way arrows are used to represent the edges or "links" in the structure. This follows the conventional representation of digraphs. This is *not*, however, intended to suggest that the links can only be traversed in one direction, nor that the zzstructure should be implemented with one-way pointers. Indeed, as should become clear, links in zzstructures should be traversable in either direction.

2.2. A list-oriented interpretation

One way to think of a zzstructure is as a set of nodes together with a set of lists of nodes. Each list has a corresponding (not necessarily unique) colour associated with it, and each node may appear in at most one list of each colour. Thus, each list gives the nodes forming a path or cycle whose edges are of the list's colour.

The zzstructure in Figure 1 could be described with the following lists. We prefix each list with its colour.

- Red: J, H, A
- Red: B, F, I, E, B
- Red: C, G, M, L
- Green: D, A, B, F, C
- Green: J, H, E, I, G, M
- Blue: B, I, K
- Blue: C, F, L, G, C

Each list corresponds to what Nelson calls a *rank* of cells [16]. Ranks may be traversed in a *posward* (with the directed edges) or *negward* (against the directed edges) direction [18]. If the rank is not a cycle (i.e. not a *ringrank* [21]), then it has a negward-most cell called the *headcell* [21].

Indeed, according to [27],

> "A slightly different way of looking at the structure that may sometimes helps thinking about it is to consider dimensions as {\em labeled lists} of cells. That is, instead of considering cells and connections, consider lists (each list labeled with a string) of cells where the same cell may be on several lists (but only one with any given label). It is not difficult to see that this is exactly the same structure as above but viewed from a different angle: emphasizing the ranks instead of the single cell and its connections."

Here, each "label" is an edge colour.

Nelson claims [14] that zzstructures are a generalization of his 1965 zipper lists [4].

See Dave et al. [2] for an example of related work in path-centric browsing.

Note that zzstructures could be equivalently defined as a kind of hypergraph [1], where each hyperedge corresponds to one list (or *rank*), and each hyperedge is an *ordered* tuple of nodes, and hyperedges of the same colour may not intersect.

2.3. A space-oriented interpretation

Let N be the number of edge colours in a zzstructure Z. We can think of Z being embedded in an N-dimensional manifold, where each edge colour corresponds to a different spatial *dimension*. Thus, travelling from a node to an adjacent node along an edge of colour C corresponds to moving through the manifold along the direction corresponding to *dimension* C. The edges in a zzstructure can act as "wormholes" that "teleport" someone traversing the structure to arbitrary nodes. Thus, the manifold in which the zzstructure is embedded is not, in general, Euclidean.

Perhaps not surprisingly, other generic names that have been proposed for ZigZag are "zzspace" and "quantum hyperspace" [15].

2.4. Other descriptions of zzstructures

Analogies used to describe zigzag:

> "The ZigZag system [...] resembles [...] perhaps a spreadsheet cut into strips and glued into loops." [15]
>
> "It's best compared to beads on a string: every cell is like a bead with many holes, and you can put one string of any color (dimension) through each bead." [14]
>
> "there is no way I can explain it other than with this visualization: imagine tiny cells of addressable information existing in a large container. Then, imagine taking needle and thread and stitching together into a string (a dimension!) whatever cells you like. Then imagine another dimension stitching together more cells (and some of those already captured in another dimension). Do this for a while and you have a massive, multidimensional representation of whatever those cells are about." -- http://xanadu.com.au/mail/zzdev/msg01566.html

The last two analogies are appropriate, so long as one keeps in mind that there may be *multiple* strings of each colour (i.e. multiple *ranks* along each *dimension*), but at most one string of each colour may pass through a given bead or cell (i.e. each cell appears in at most one rank along each dimension).

In keeping with the analogy of tangled strings or threads, one can imagine choosing a single thread, and straightening it by pulling it taut. The effect would be to "flatten" the structure along one rank. By successively straightening different threads, one may obtain very different rearrangements of the nodes.

---

3. 2D Cursor-centric viewing of Zzstructures

Zzstructures can, in general, be very large and complicated, having many nodes and dimensions (edge colours). This makes it difficult to draw them in a clearly understandable way. One strategy for drawing is to pick a single node as a focal point, and draw the neighbourhood around that node. Nelson [14,17] describes a 2D cursor-centric viewing scheme that makes use of 2 spatial dimensions at a time, and locally "flattens" a subset of the neighbourhood around a cursor (i.e. a selected node). More specifically: we find the subset of nodes that are connected to the cursor's node via edges along the two chosen dimensions. This subset of nodes can be embedded in a (non-Euclidean) 2D manifold, which is then displayed in a flat, 2D view. Wormholes are handled, in part, by displaying virtual copies of nodes. By changing the chosen pair of dimensions, we can

visually reveal, hide, and rearrange nodes in interesting ways (see Nelson [14] for a step-by-step example). Multiple arrangements (lists, 2D arrays, trees, ...) can co-exist by using different dimensions to encode each arrangement.

The 2D cursor-centric view is perhaps the earliest or most basic drawing scheme that has been proposed for zzstructures, and detailed examples of it are given below. Other, more sophisticated visualizations, such as a "vanishing" view, have been implemented and are described at [26,27].

The examples in this section are somewhat contrived. The graphs have only 5 to 9 nodes, whereas most of the 2D views have room for 5x5=25 nodes, which is more than enough to show the entire graph. Thus, there is much redundancy in many of the 2D views. These examples were of course chosen to be as simple as possible and still illustrate the desired concepts. Keep in mind that, in practice, the zzstructure being viewed may be very large, and there is usually *not* enough room in the 2D view for all of the nodes.

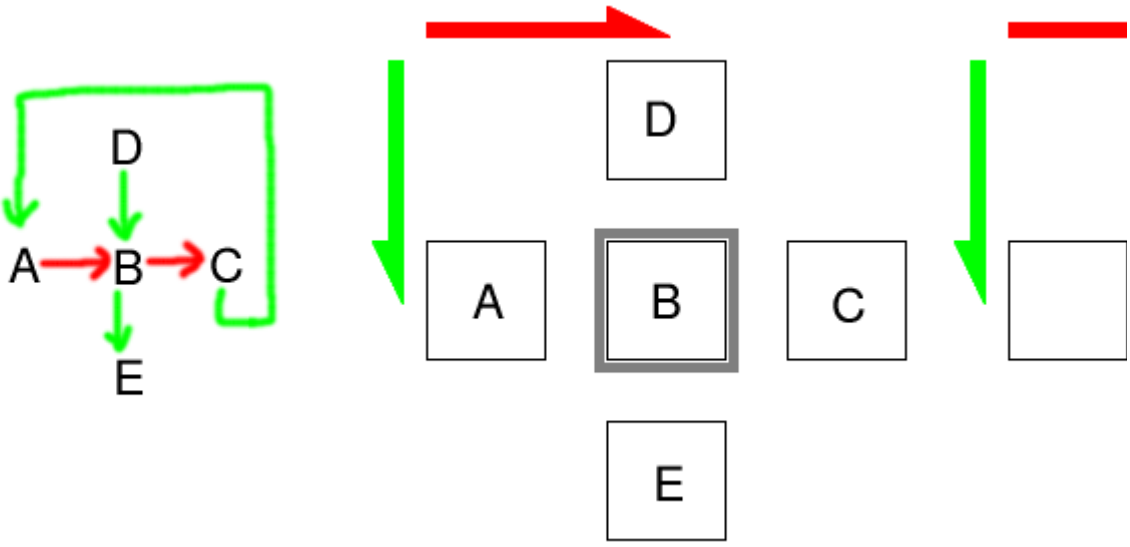

Figure 2. Left: a simple zzstructure. **Middle:** a 2D view, with the cursor (in grey) centred at node B, showing the immediate neighbours of B along the red and green dimensions. **Right:** a 2D view showing the immediate neighbours of node A. The way that C appears "above" A shows how edges in the zzstructure can act as spatial "wormholes".

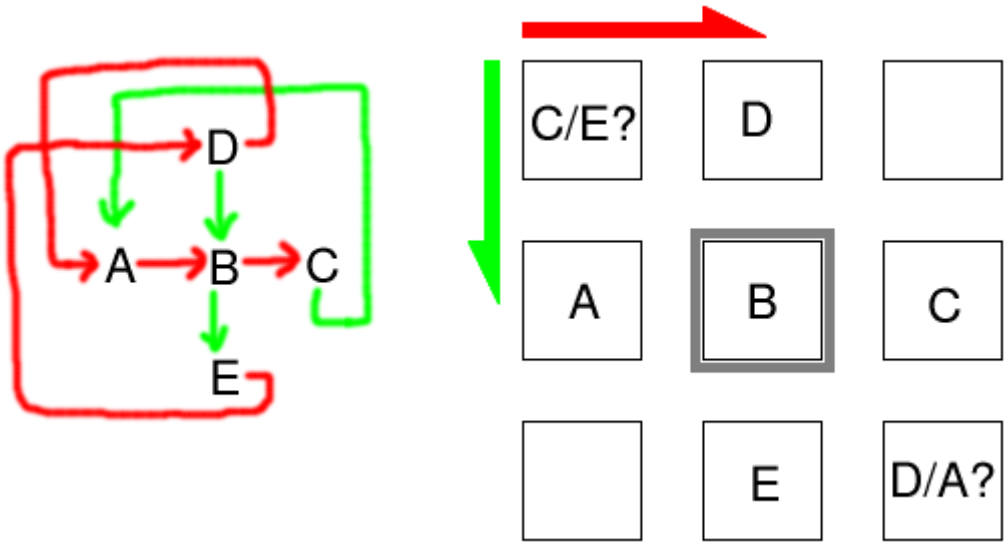

Figure 3. Left: a slightly more complicated zzstructure than the one in Figure 2(left). **Right:** a 2D view of B's neighbours. Here we see a basic problem when trying to "flatten" more than a cell's immediate 4 neighbours. In the upper-left cell, should we show C or E, these being respectively the node "above" A and the node to the "left" of D ? Similarly, in the lower-left cell, should we show D or A, these being respectively the node to the "right" or E and the node "below" C ?

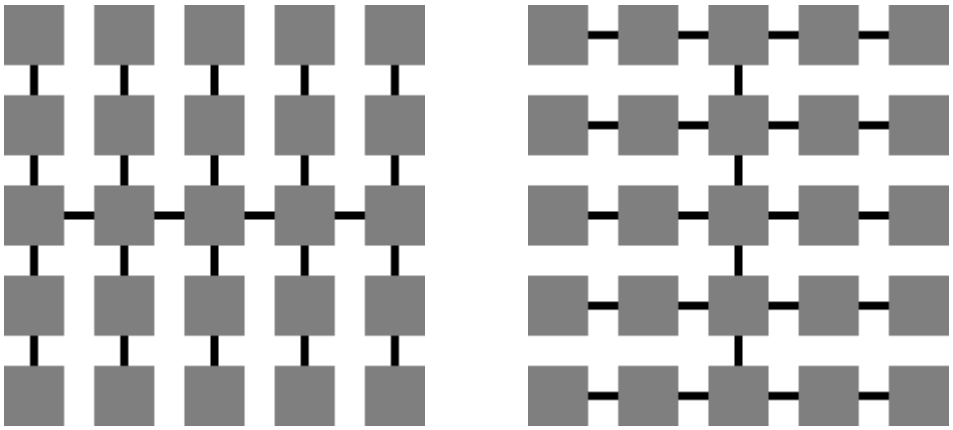

Figure 4. A solution to the problem illustrated in Figure 3(right). When showing a 2D view of a node's neighbourhood, we extract a substructure that can be flattened easily. Only nodes that are connected to the cursor in one of the 2 ways illustrated above are shown. **Left:** the column view, or *H-view* [17], so named because of its resemblance to the bars in the capital letter H. **Right:** the row view, or *I-view* [17], so named because of its resemblance to the bars in the capital letter I (with serifs).

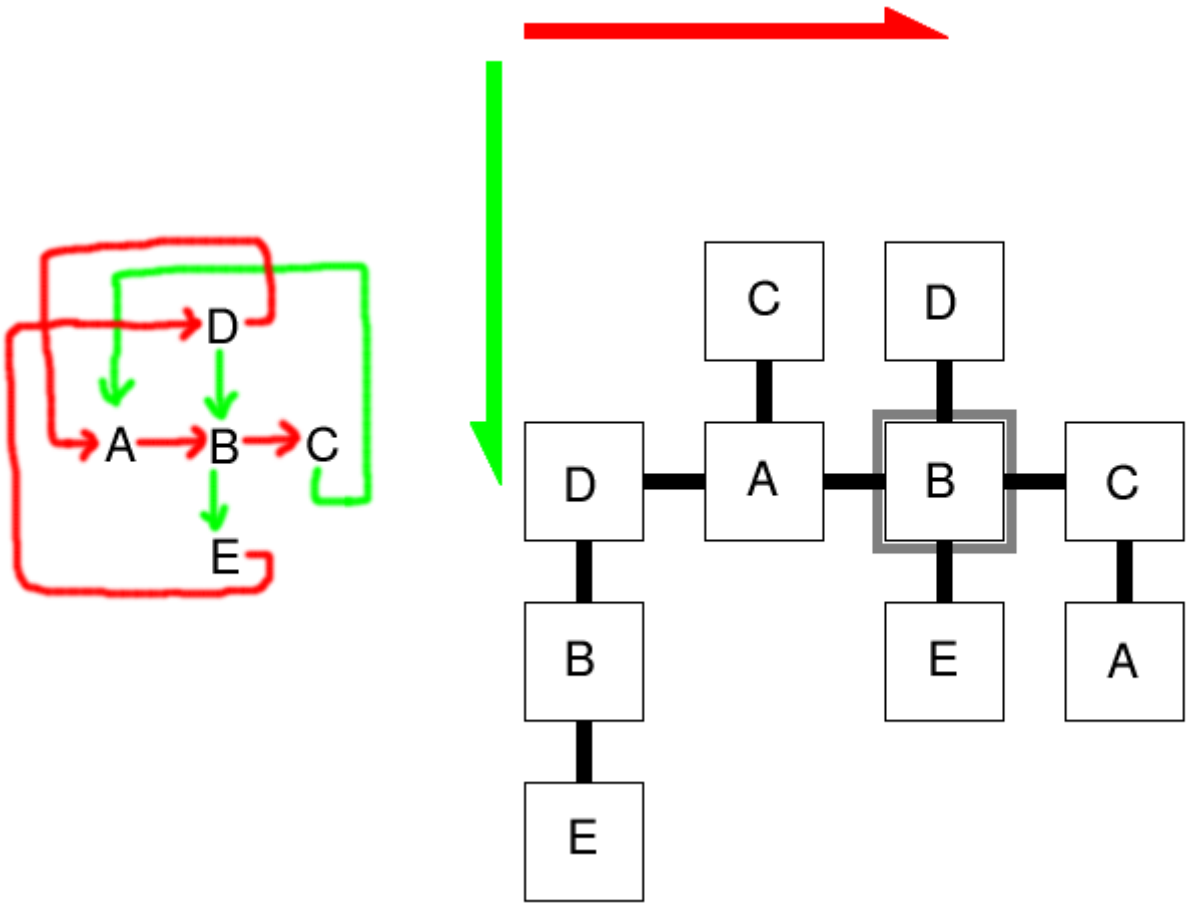

Figure 5. Left: the zzstructure from Figure 3(left). **Middle:** an H-view centred at B. **Right:** an I-view centred at B.

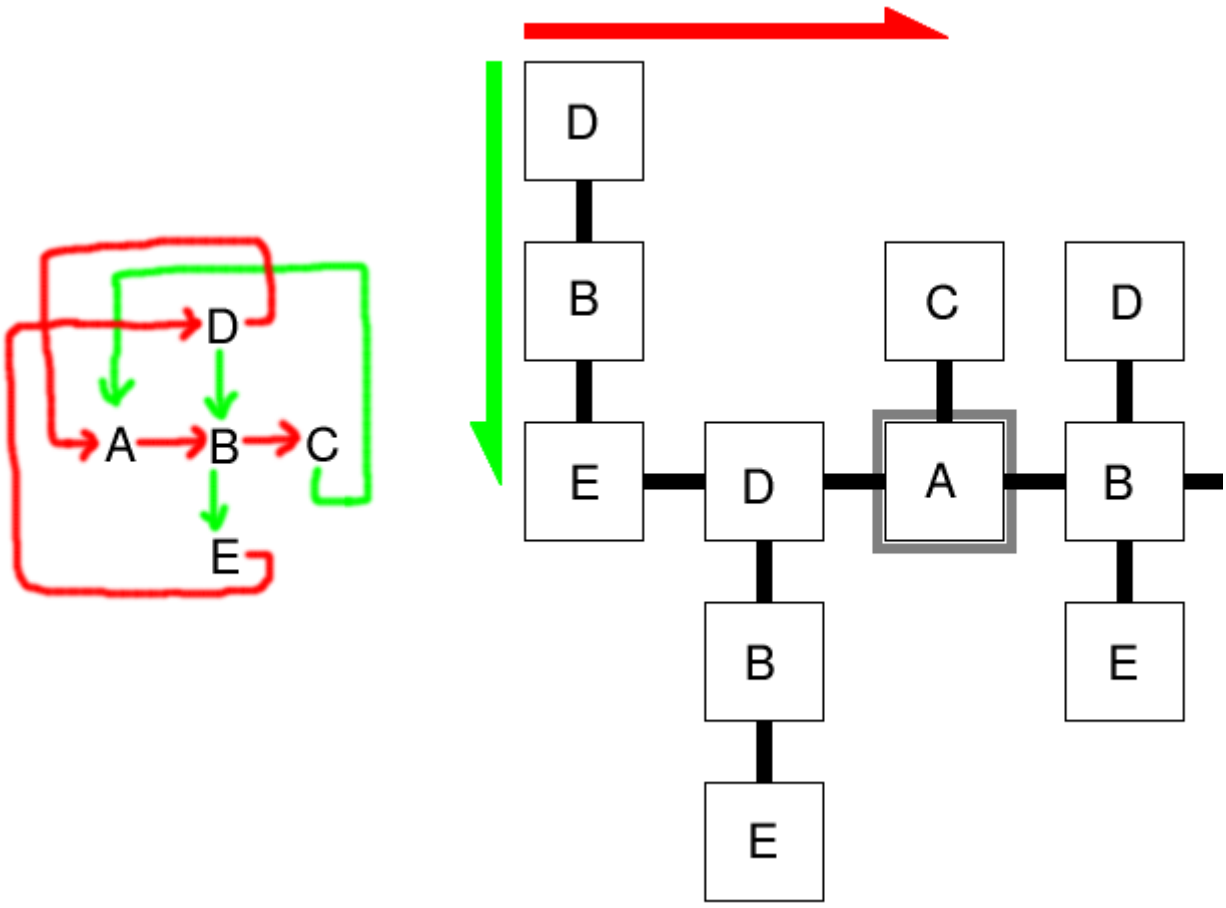

Figure 6. Left: the zzstructure from Figure 3(left). **Middle:** an H-view centred at A. Notice how this appears to be Figure 5(middle) scrolled sideways by 1 column, since the cursor has moved sideways from B to A, and the columns extracted by the H-view (Figure 4(left)) are mostly the same. **Right:** an I-view centred at A. Notice how this does *not* appear to be Figure 5(right) scrolled sideways by 1 column. The cursor has moved sideways from B to A, and this causes *different* rows to be pulled out by the I-view (Figure 4(right)).

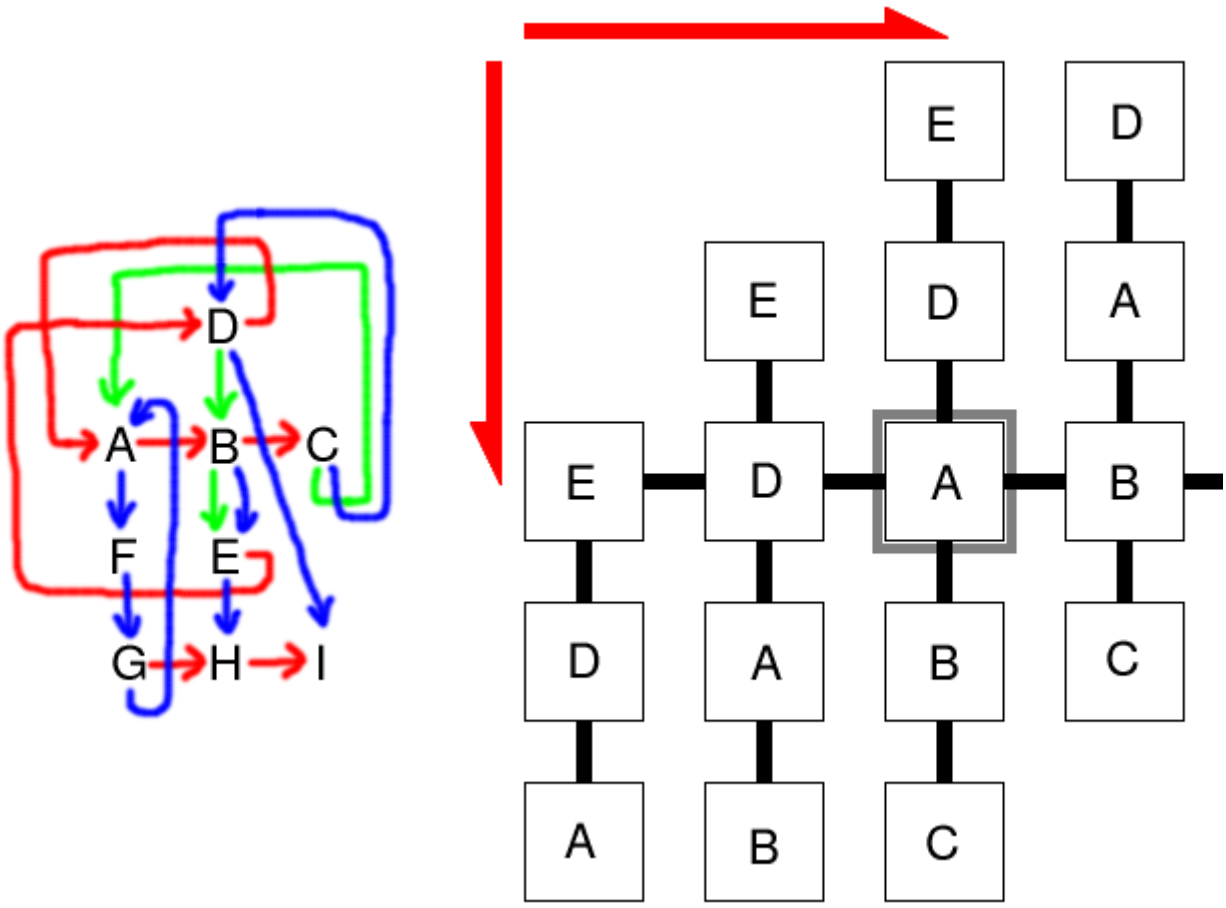

Figure 7. Left: a more complicated zzstructure, involving 3 edge colours. 2D views require choosing just 2 of the edge colours at a time to act as the 2 dimensions of the view. **Right:** an H-view where the red dimension has been mapped to both x and y in the 2D view. Although this 2D view is not useful, it illustrates the elegance and consistency of the concepts involved. Nelson gives a similar example of one dimension being mapped to both axes of a 2D view [14].

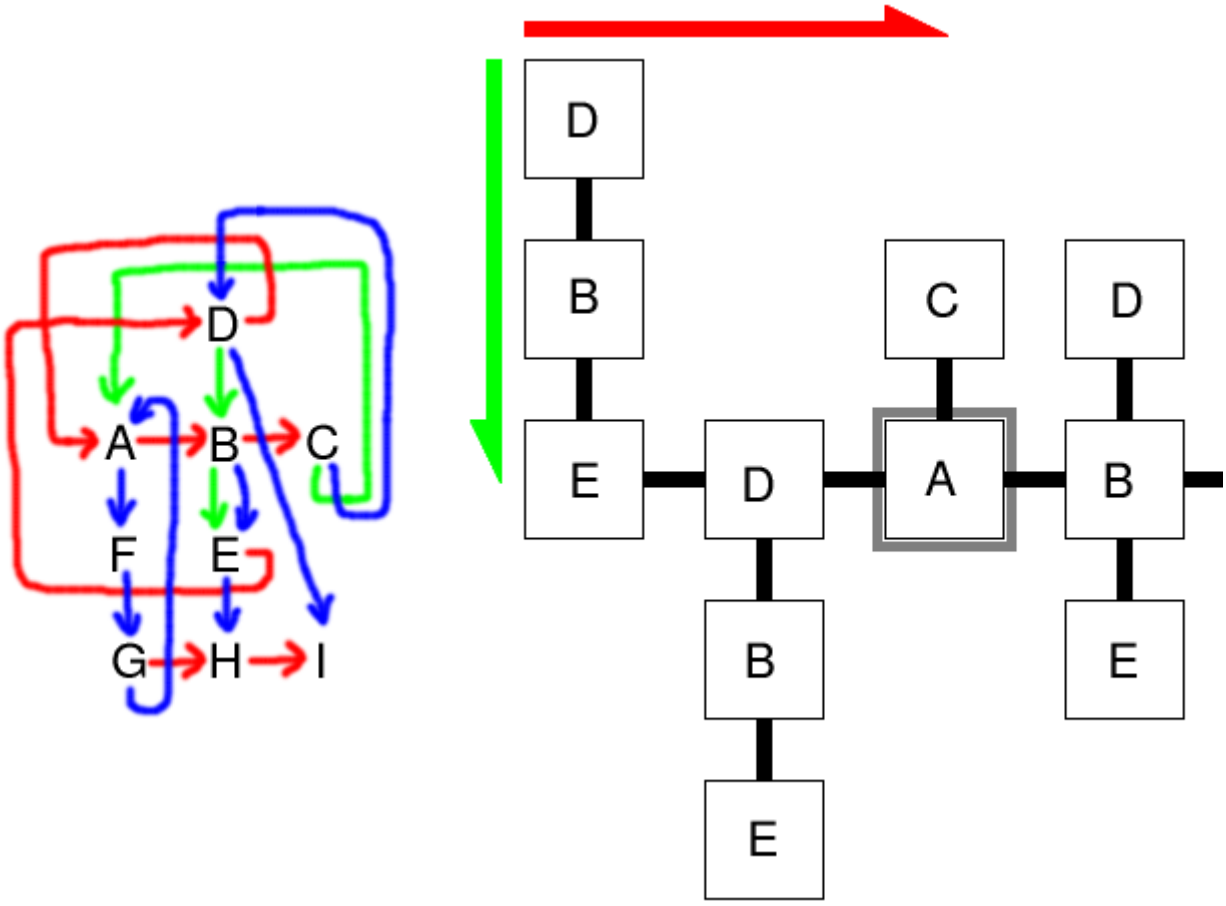

Figure 8. An example of how the view changes when the colour-axis mapping is changed, without changing cursor position. **Left:** the zzstructure from Figure 7(left). **Middle:** an H-view centred at A, which happens to be identical to Figure 6(middle). **Right:** the same H-view centred at A, where the red dimension is still x, but now the blue dimension is y. We also see here two possible enhancements to the basic 2D view. First, although the nodes displayed were chosen strictly based on the H-view template (Figure 4(left)), also shown are "incidental edges" (in grey) where adjacent cells happen to appear in neighbouring positions. These "incidental edges" are not part of the H-view template, but make the visualization more informative. Second, ellipses indicate directions in which nodes have neighbours that don't fit in the 2D view, or that weren't extracted by the H-view. This is an example of elision.

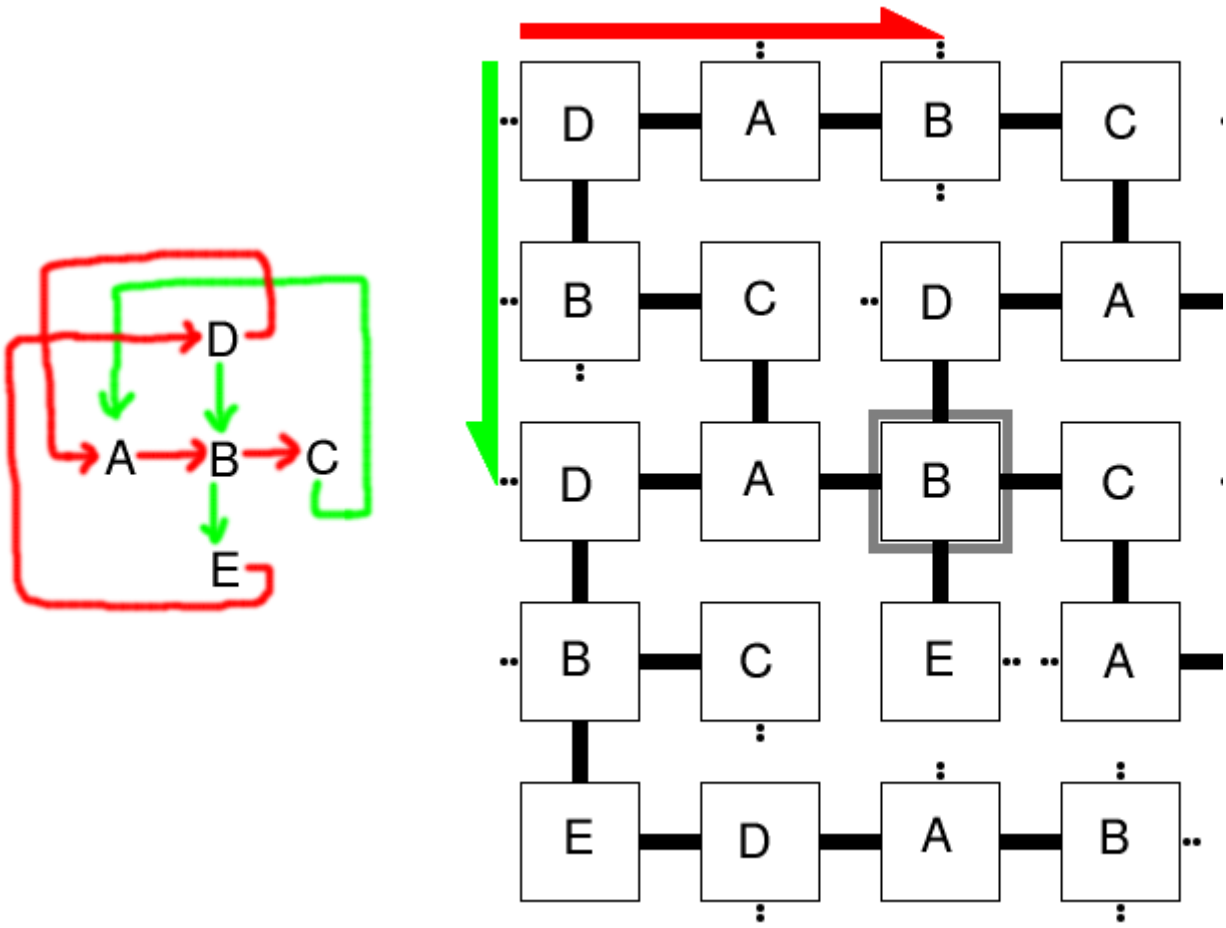

Figure 9. H-views and I-views do not, in general, make use of all the space available in a 2D view. I propose H^+- and I^+-views, where we first extract all the nodes possible using the traditional H- or I-view, and then try to fill any remaining space by extracting any other neighbouring nodes available. **Left:** the zzstructure from Figure 3(left). **Middle:** an H^+-view centred at B. This is an augmentation of Figure 5(middle). (Note the interesting cycle through the cursor formed by the nodes B, D, A, C, B, A, D, B, C, A, B. Mathematically, one may wonder which cycles in a graph might cause one to cycle back to the same spatial position in the 2D space of the view. The edges in the cycle are (B,D), (D,A), (A,C), (C,B), (B,A), (A,D), (D,B), (B,C), (C,A), (A,B). We can reorder this list, pairing up the edges as (B,D), (D,B); (D,A), (A,D); (A,C), (C,A); (C,B), (B,C); (B,A), (A,B); which shows that every edge involved is traversed twice, once in each direction. This begins to explain how the (red,green) spatial coordinates of the nodes cycle back to the same position.) **Right:** an I^+-view centred at B. This is an augmentation of Figure 5(right).

---

4. Example Zzstructures

Nelson states [16,19] that zzstructures generalize lists, cycles, tables (a.k.a. grids, spreadsheets, or 2D arrays), and trees. In the below examples, we illustrate these and other cases.

Keep in mind that zzstructures also support any *simultaneous combination* of the below structures, since each substructure can be encoded using different colours. For example, a set of nodes corresponding to files could be organized in a list using one dimension, and in a 2D tabular grid

using two other dimensions, and in a tree using additional dimensions. Mapping different dimensions to the 2D view allows the user to rearrange the nodes and see each substructure.

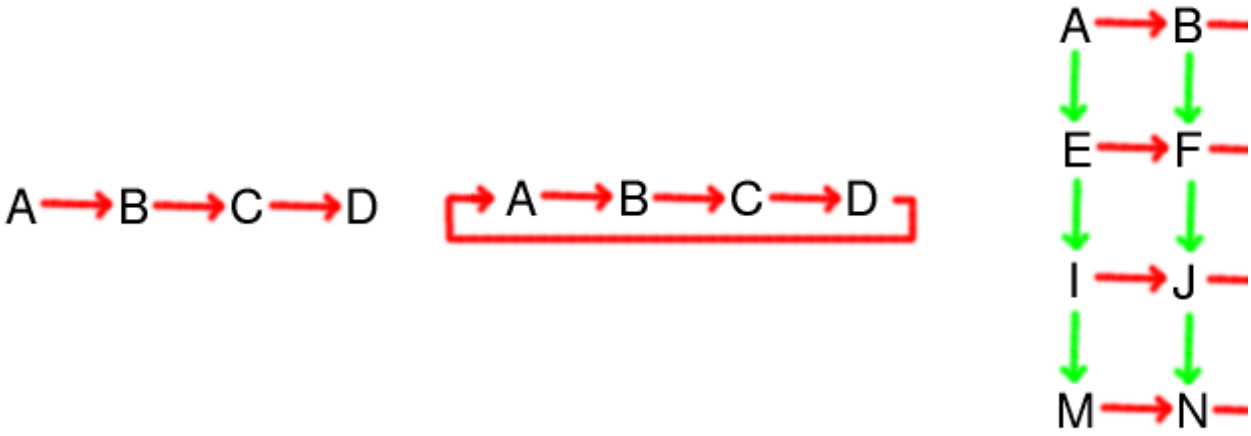

Figure 10. A list, cycle, tabular grid, and torus, encoded as zzstructures. Exercise for the reader: what difference, if any, is there between viewing these structures with an H-view and an I-view ?

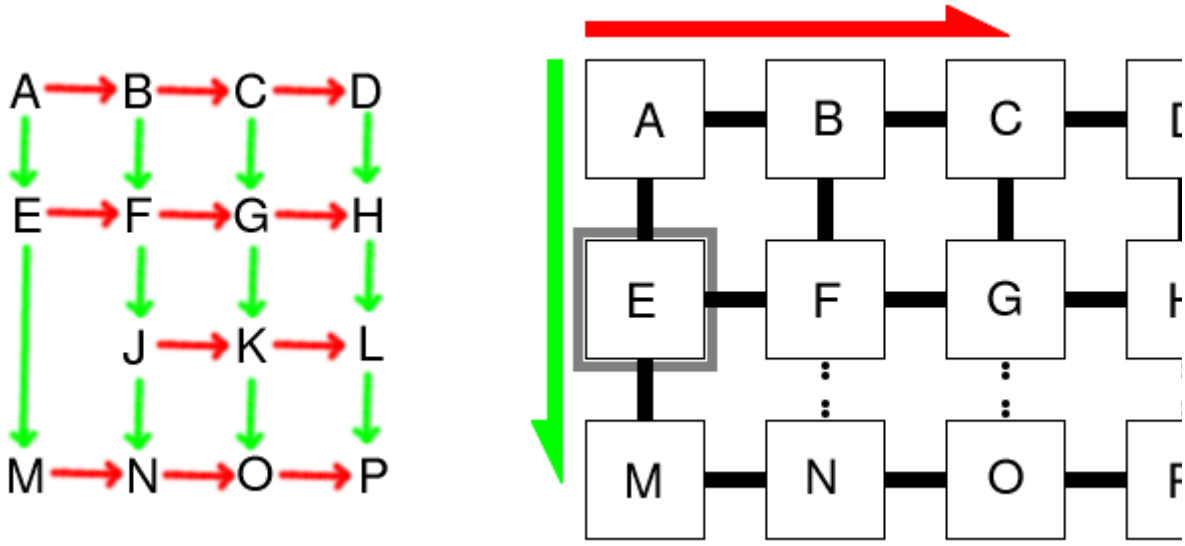

Figure 11. This example demonstrates how rows (or columns) can be hidden in a 2D view, which can be useful for collapsing or folding away information. (This can be compared to the behaviour of a text outlining environment, that allows sections and subsections of text to be collapsed or expanded.) Nelson gives a similar example of a hidden row [20]. The hiding of multiple rows would enable, for example, folding away so-called *sandwich subroutines* [22] in a cellular programming language. **Left:** a zzstructure. **Right:** an I-view of the zzstructure with the cursor at E. The 3rd row of nodes J, K, L is hidden. Moving the cursor to F, or switching to an H-view, would reveal the 3rd row.

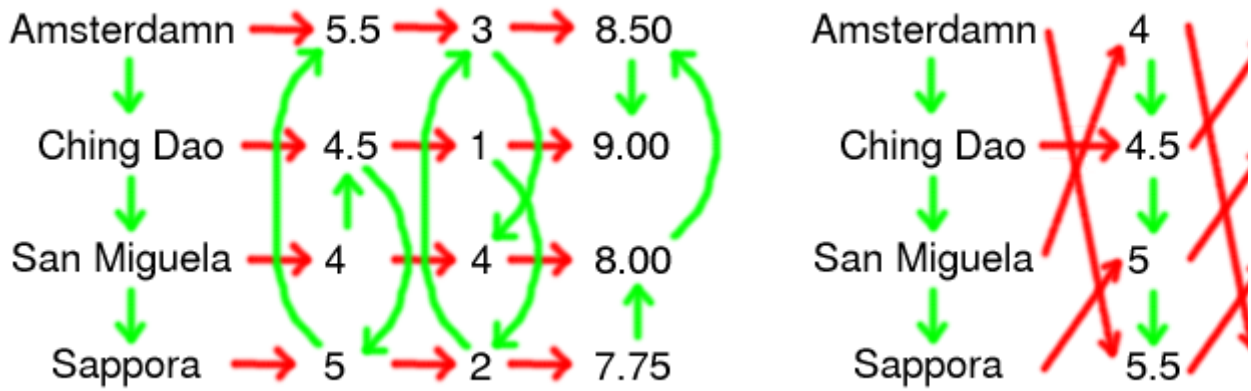

Figure 12. This zzstructure encodes a table of statistics for 4 fictional beers, and is drawn here in two different ways. In both cases, the 4 columns are beer name, alcohol percentage, personal ranking, and price in dollars per six-pack; and the nodes in each column are linked with green edges in alphabetical or numerical order of their entries. At left, nodes are arranged such that rows appear in straight lines, and at right, nodes are arranged such that each column appears in sorted order. Under an I-view, the rows would appear in straight lines, and the column through the cursor would be sorted. Under an H-view, the columns would each be sorted (though they would also be shifted vertically with respect to each other), and the row through the cursor would appear in a straight line. Thus, an I-view allows us to sort the rows by any single column, and an H-view allows us to see the ranking of any single row within each column. The zzstructure here is an example of *twisted rows*. Nelson gives a similar example of *twisted columns* [20].

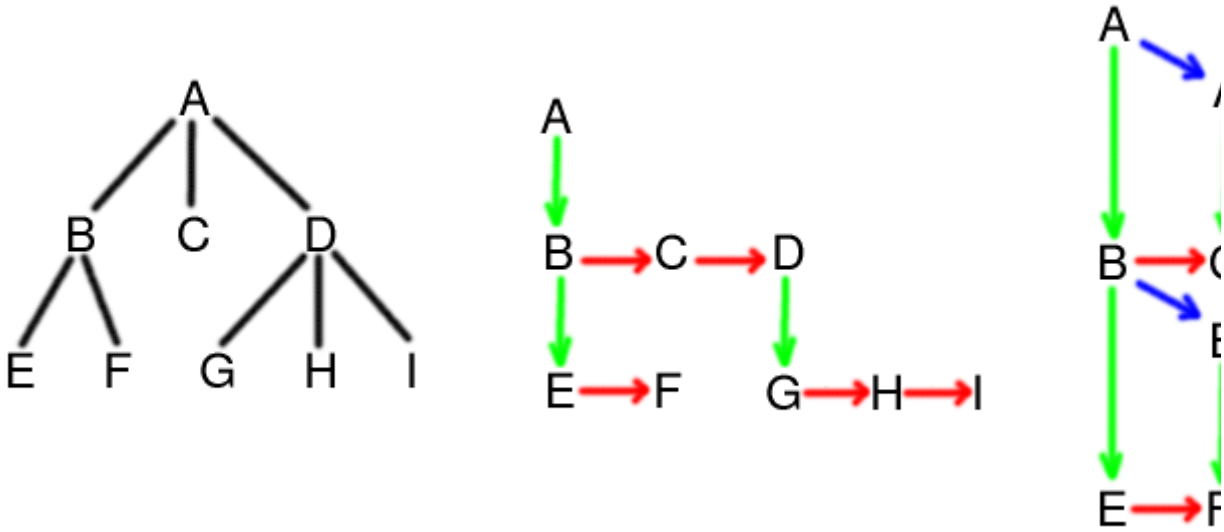

Figure 13. Tree structures require many-to-one links between parent and child nodes. However, our definition of zzstructures (specifically, restriction R) appears to preclude many-to-one links in zzstructures. All the same, it turns out that we can encode trees in zzstructures [27,19], but at the cost of using more than one edge colour. **Left:** a traditional tree. **Middle:** encoding the tree as a zzstructure. This encoding is analogous to the left-child, right-sibling pointer implementation for tree data structures [1]. Navigating from a child node to a parent node requires first navigating to the left-most child of the parent, and then following the edge up to the parent. **Right:** an alternate

encoding of the tree in a zzstructure. Many-to-one links can be simulated using a 3rd dimension (blue) which might be called a "clone" dimension. This way, navigation from a child to a parent (or clone of the parent) can be done in one step, by following the edge upward.

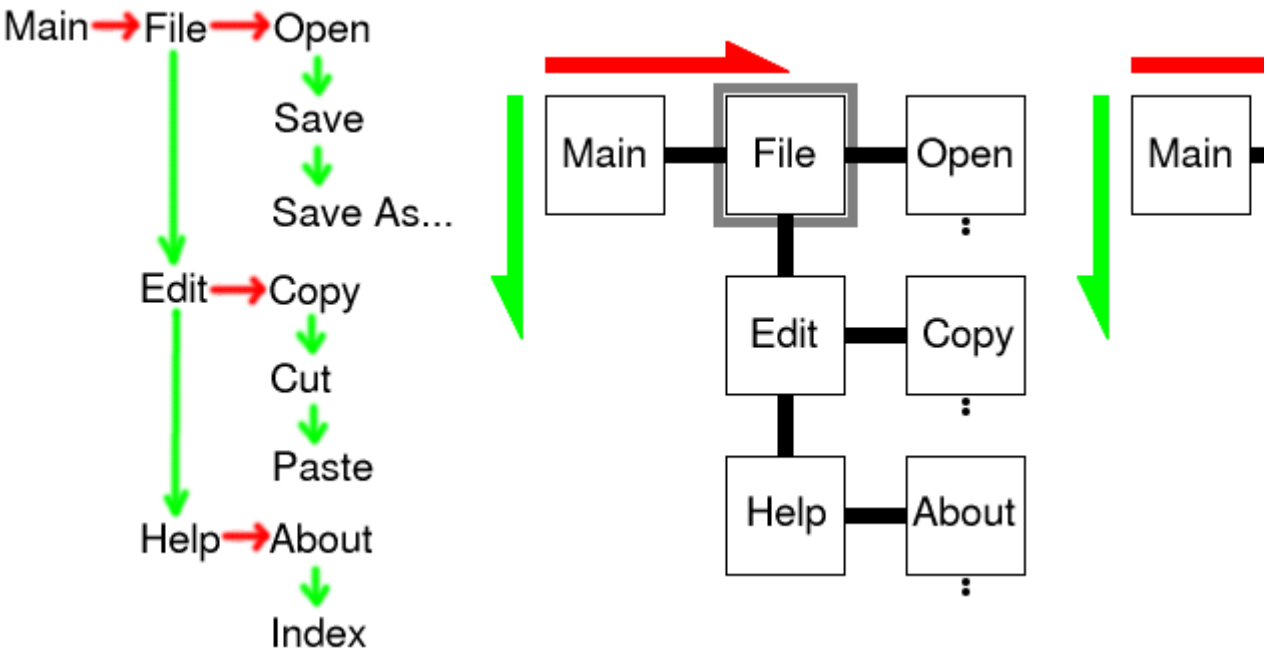

Figure 14. This figure demonstrates navigation of a zzstructure-encoded tree with a 2D cursor-centric view. Different subsets of nodes are hidden by different views, in the same way that rows and columns can be hidden as shown in Figure 11. (Interestingly, the menu-like behaviour was not "built-in" to the 2D cursor-centric view scheme, but is rather a side-effect of using H- and I-views.) **Left:** a hierarchical menu of commands implemented as a zzstructure, in the style of Figure 13(middle). **Middle left:** an I-view with the cursor at the File node shows all the menu items at the same level as File. To see menu items under File, the user must either switch to an H-view, or move the cursor to the Open node. **Middle right:** an H-view with the cursor at the File node shows all the menu items under File. **Right:** an I-view with the cursor at the Open node.

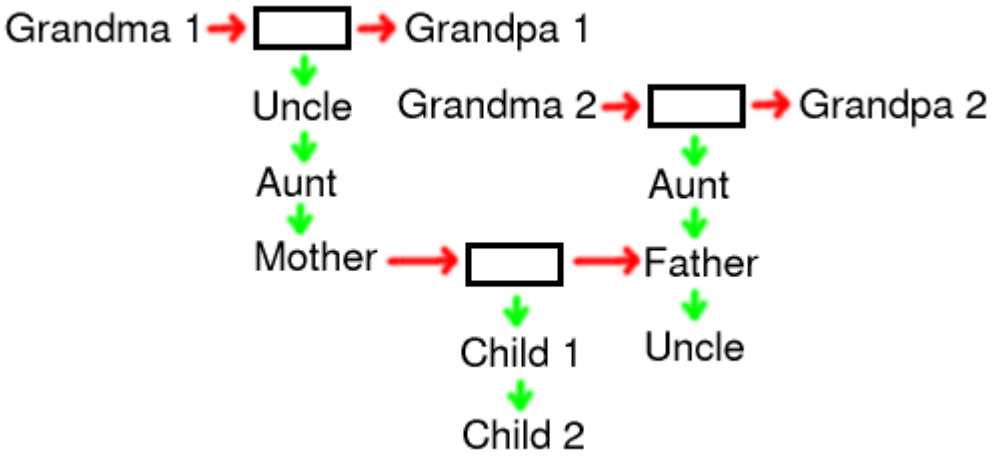

Figure 15. A family tree encoded as a zzstructure [26]. The two dimensions are spousal (red) and children (green). The empty rectangles are "marriage" or "reproduction" nodes. Children could be

ordered by birth. Multiple spouses could be modelled by "cloning" individuals along a 3rd dimension (see Figure 13(right)). This graph could, of course, be extended indefinitely, and a 2D cursor-centric view is only one way of locally flattening the family tree for viewing. (After showing a similar family tree example, Nelson has remarked "we did not create a genealogy program" [26], pointing out that zzstructures and the 2D cursor-centric view are flexible enough to enable modelling and navigation of many rich structures.) Exercise for the reader: what difference, if any, is there between viewing this family tree with an H-view and an I-view ?

---

5. Relationship between Zzstructures and other kinds of graphs

As already shown in Figures 11, 12, and 14, the nodes of a zzstructure can be revealed, hidden, or visually rearranged in interesting ways by moving the cursor or by changing the type of view in effect (e.g. changing an H-view to an I-view). Another action used when navigating a zzstructure is to change the pair of dimensions mapped to the 2D view (e.g. Figure 8, or see Nelson [14] for a step-by-step example). As already mentioned, multiple arrangements (lists, 2D arrays, trees, ...) can co-exist in a zzstructure by using different dimensions to encode each arrangement. Each arrangement is then accessible by mapping the appropriate dimensions to the 2D view.

> "ZigZag allows you to arrange the *same* things into *different* traditional structures" [27]

The ability of zzstructures to encode multiple arrangements comes essentially from having edges that are *coloured*, allowing each edge to be marked as belonging to one arrangement or another.

> "Having dimensions is essential to having any structure in the space: they allow us to build substructures. In a generic graph you can't define easily how for example lists or hierarchies can be represented. In ZigZag it's easy: a list is a rank along a dimension, and hierarchies are composed by having a cell's children on a rank along one dimension and connecting the rank (ie. the headcell of the rank) to the parent along another dimension.
>
> Since dimensions allow substructuring, they also allow building custom views. The tree view [...] is a good example: it renders the hierarchical substructure mentioned above as a nice tree. The presentation view [...] is another, possibly better example." -- http://www.advogato.org/article/156.html

This makes zzstructures more powerful and more general than graphs with untyped edges. In particular, zzstructures are more general than multitrees [3], and even more general than polyarchies [7,8,9].

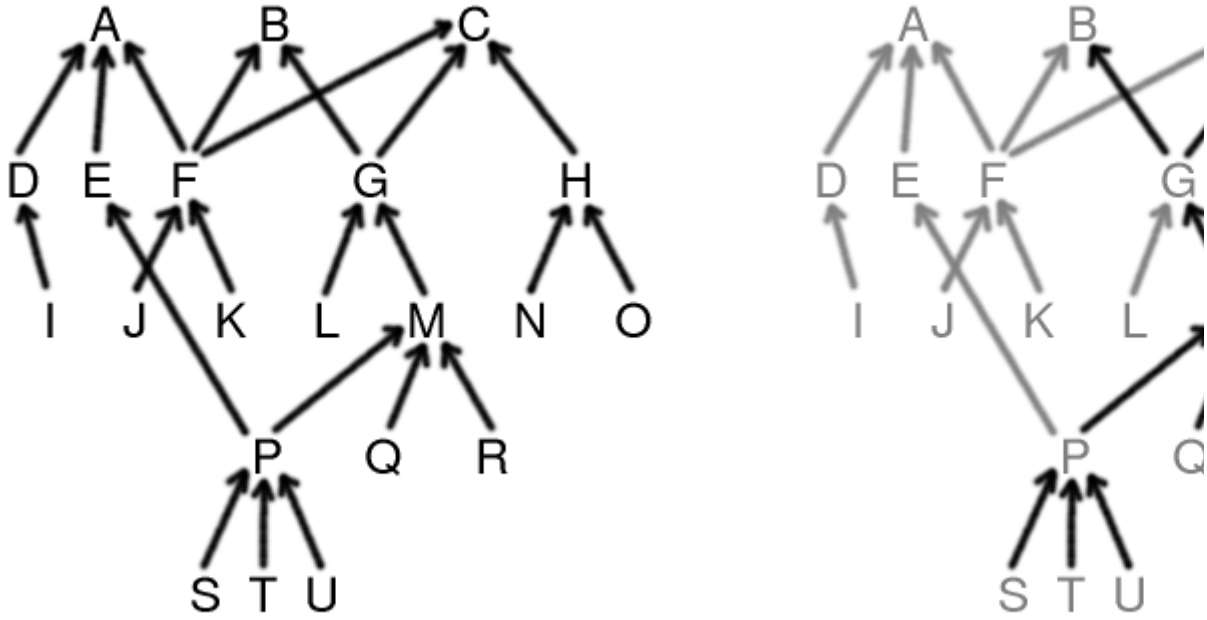

Figure 16. Multitrees [3] are a kind of DAG (directed acyclic graph) where overlapping trees share subtrees. The trees in a multitree are implied by the multitree's structure, and thus need not be explicitly marked or coloured. Each node in a multitree has a tree of ancestors and a tree of descendants. **Left:** a multitree. **Right:** Node M is highlighted, along with its tree of ancestors and its tree of descendants.

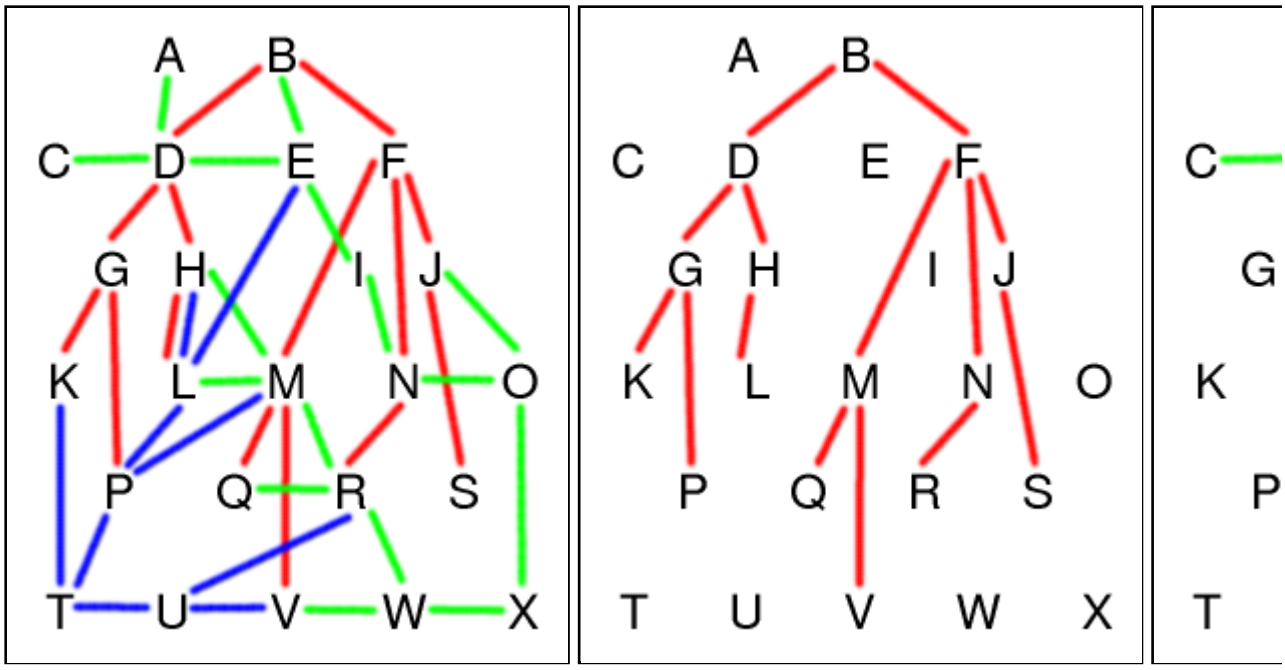

Figure 17. A polyarchy [7,8,9] is a set of trees which may share arbitrary nodes. The trees in a polyarchy must be somehow explicitly marked or coloured, hence it is convenient to define a

polyarchy as a multigraph with coloured edges -- one colour for each tree. At left is a polyarchy involving 3 trees, and the other images show each tree. Note that, since there is no indication of which nodes are the roots of the trees, these are *free* trees [1]. However, if the edges were directed toward the root of each tree, or if there were some other information to indicate the roots, we would have *rooted* trees [1]. Hence, a polyarchy can be thought of as a directed or undirected multigraph with coloured edges.

Robertson et al. claim that multitrees are a subset of polyarchies ("MultiTrees are multiple hierarchies with shared subtrees. But polyarchies are multiple *intersecting* hierarchies, sharing at least one node rather than sharing subtrees. Hence, MultiTrees are a subset of polyarchies." [8 and 9]). This is roughly true, since the trees in a multitree may only share subtrees, whereas the trees in a polyarchy may share arbitrary nodes. However, as seen in this section, multitrees are a kind of directed graph that have no need for coloured edges, because the trees in them are implied by the multitree's structure. Polyarchies, on the other hand, must have their trees explicitly marked, and so are conveniently defined as a kind of (directed or undirected) multigraph with coloured edges. Since these are different kinds of objects, multitrees are not technically a *subset* of polyarchies. However, the information contained in a multitree can always be encoded with a polyarchy, and in this sense polyarchies do generalize multitrees.

Both multitrees and polyarchies (as defined here) make use of many-to-one links between nodes, which are not allowed in zzstructures. However, Figure 13 shows how many-to-one links, and hence trees, can be "simulated" or encoded in zzstructures, by using additional colours. It follows that the information contained in a multitree or polyarchy can always be encoded with a zzstructure. In addition, and unlike multitrees or polyarchies, zzstructures may contain arbitrary lists, tabular grids, trees, and other structures, all simultaneously.

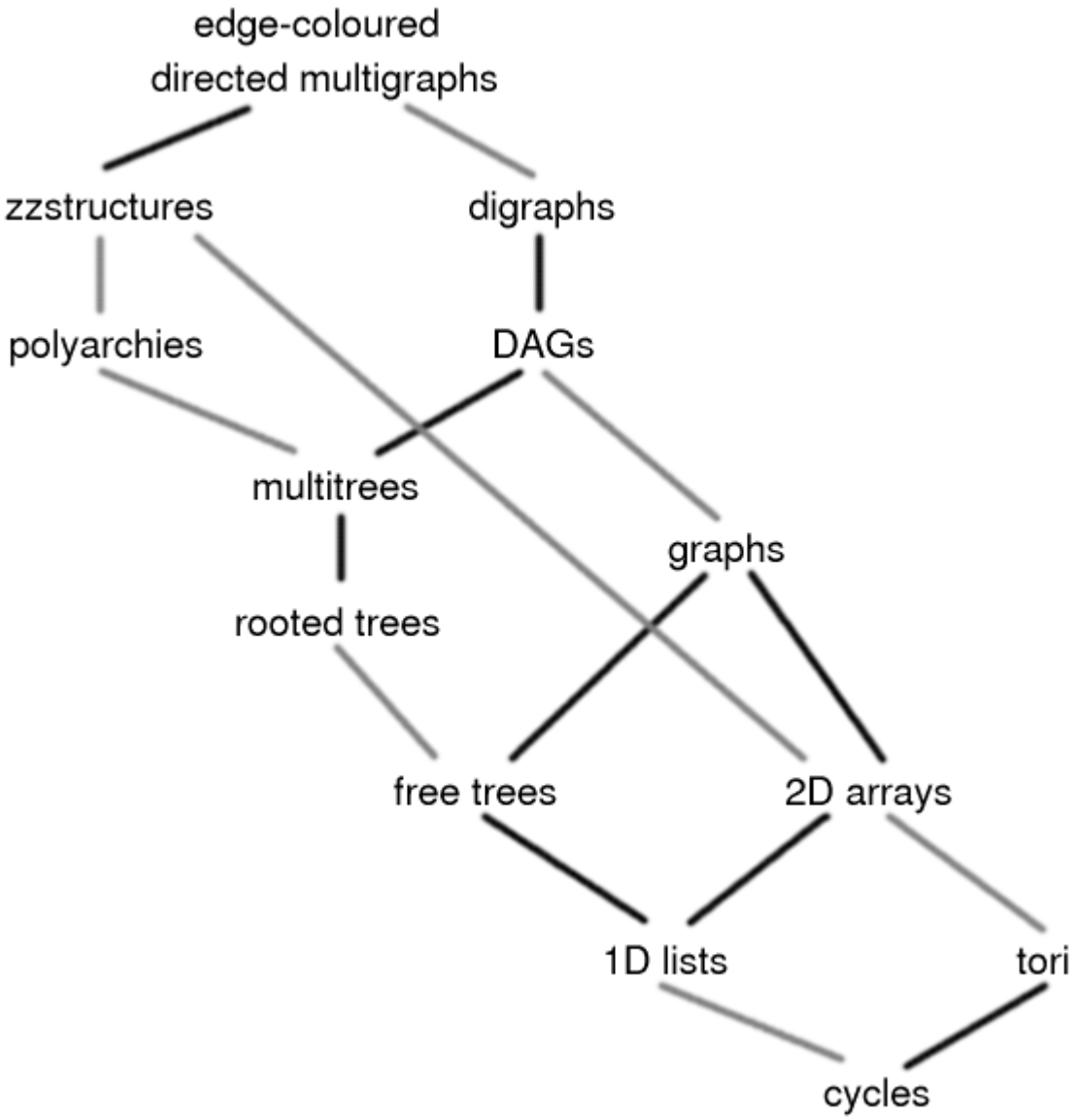

Figure 18. A tentative subsumption diagram of selected data structures. Structures that appear higher in this diagram are more general and subsume structures below them. "Digraphs" refers to directed graphs; "polyarchies" are assumed to be of *rooted* trees; "graphs" refers to simple, undirected graphs; "free trees" refers to trees with no node singled out as a root; "2D arrays" refers to grids or spreadsheet-like tables; "tori" refers to wrap-around 2D arrays (i.e. arrays with no boundaries); "cycles" refers to wrap-around 1D lists (i.e. lists with no ends). Black lines denote superset-subset relationships (e.g. multitrees are a subset of DAGs). Grey lines denote a different kind of subsumption relationship that doesn't precisely involve supersets and subsets (e.g. a free tree is not a kind of rooted tree, hence free trees are not a *subset* of rooted trees, however a free tree can be encoded as a rooted tree *with no loss of information*). Note that the form of this diagram depends on how the structures in it are defined precisely, and the relationships shown are not necessarily complete. For example, any digraph could arguably be encoded as a polyarchy, with no loss of information, simply by making each edge in the digraph a distinct tree in the polyarchy. However, this is rather "hackish" (inelegant), and questionable subsumption relationships like this might make the diagram more confusing than enlightening.

It is apparent from Figure 18 that we could generalize zzstructures by removing restriction R, thus allowing all edge-coloured directed multigraphs. Such structures would have the same ability as zzstructures to encode multiple, overlapping substructures, and would also afford many-to-one links using a single edge colour. Is there any value in having restriction R ?

Restriction R requires that each node have at most one incoming and one outgoing edge of each colour. As pointed out earlier, this implies that there is a unique way to move backward or forward along the path of each colour through a node. This makes it more natural to think of the links as "2-way" links: if a user arrives at a given node from a given direction, returning to the previous node is simply a matter of moving back in the opposite direction. This would not be the case with many-to-one links, where a user could arrive at the same node via the same direction from many different previous nodes. With restriction R, backing-up along an edge requires only remembering which colour was traversed. Without restriction R, backing-up requires remembering which node the user came from.

In addition, and perhaps more importantly, having many-to-one links of a single colour would make 2D viewing more problematic: in general, a node might have multiple nodes to its immediate "right" and immediate "left". The 2D cursor-centric views described in this document would not apply without adjustment. In fact, the whole spatial interpretation of zzstructures -- of edge colours as dimensions -- would seem less appropriate. Thus, there is definite value in the constraints imposed by restriction R.

---

6. Some Open Issues

Below are some open issues which occurred to me while preparing this document. I am not aware of how much work has already been done by Nelson or others to address these or other issues.

- How would zzstructures be best used for organizing files ?

  Since zzstructures subsume polyarchies, clearly files could be organized according to one or more traditional hierarchical trees. However, would it make sense to have, for example, two distinct trees that each cover the entire set of files on one's harddrive ? What if these two trees are mostly identical, and only differ in small, localized regions ? Would having multiple trees create more work for the user, since each new file might have to be explicitly added to multiple trees ?

  How can zzstructures best solve the common problem of users wanting to place a new file in multiple "folders" ? By cloning the file along a special clone dimension, and then placing each clone under a different parent node ? Or, are "folders" (i.e. trees) not the best model to use ?

  Figure 12 suggests how a set of data might be resorted by any column of data. However, if a user wants to sort data or files by multiple keys (e.g. first by type, and then by size; or, in the case of academic papers, perhaps first by conference name, then by year, then by author), it's less clear how a zzstructure might elegantly support this without a custom viewing scheme, or special support for dynamically regenerating links to resort files.

- If a user has a large set of data stored in a zzstructure, involving many different dimensions, might the management of a large set of dimensions become a problem ? In one

demonstration, Nelson suggested "You add a dimension simply by putting its name into the dimension list-- as a new cell, of course." [14] It seems doubtful that a linear list of dimensions would scale well to storing hundreds or thousands of dimensions.

- What's the best way to visualize zzstructures on a 2D screen ? How can we make full use of available screen space ?
- With traditional hierarchical trees, if a user wants to do an exhaustive visual search for something, they may do so simply with, for example, a pre-order traversal (visit each folder, and then visit each child recursively). Although such searches require a prohibitive amount of time with large trees, users do sometimes perform such searches within small folders or subtrees -- sometimes to find a file, and sometimes to remind themselves of the contents of a folder or subtree. However, with zzstructures, as with even simple graphs, there doesn't seem to be any straight-forward traversal that will visit every node once and be easy to keep track of. Might nodes become lost or forgotten, because they are located along some seldom-used dimension which the user never thinks to travel along ? Perhaps the user should be able to view all neighbouring nodes, along any dimensions, within a certain radius of a focal node.
- How might we reveal *all* the dimensions along which a cell has connections, especially when only a subset (e.g. 2) of the dimensions are currently being used to arrange nodes on the screen ?

References

1.
```
@book{cormen,
  author={T. H. Cormen and C. E. Leiserson and R. L. Rivest},
  title={Introduction to Algorithms},
  year=1990
}
```

2.
```
@inproceedings{dave2003,
 author = {Pratik Dave and Unmil P. Karadkar and Richard Furuta
    and Luis Francisco-Revilla and Frank Shipman and
    Suvendu Dash and Zubin Dalal},
 title = {Browsing intricately interconnected paths},
 booktitle = {Proceedings of the fourteenth ACM conference
           on Hypertext and hypermedia},
 year = 2003,
 pages = {95--103}
}
```

3.
```
@inproceedings{furnas1994,
 author = {George W. Furnas and Jeff Zacks},
 title = {Multitrees: enriching and reusing hierarchical structure},
 booktitle = {Proceedings of ACM CHI 1994 Conference on Human Factors in
 year = 1994,
 pages = {330--336}
}
```

4.
```
@inproceedings{nelson1965a,
  author = {Theodor H. Nelson},
  title = {A File Structure for the Complex,
         the Changing and the Indeterminate},
  booktitle = {Proceedings of ACM National Conference},
  year = 1965
```

```
}
```

5.
```
@article{nelson1999a,
 author = {Theodor Holm Nelson},
 title = {Xanalogical structure, needed now more than ever:
      parallel documents, deep links to content, deep versioning,
      and deep re-use},
 journal = {ACM Computing Surveys},
 year = {1999},
 month = {December},
 volume = {31},
 number = {4es},
 pages = {33},
 publisher = {ACM Press}
}
```

6.
```
@inproceedings{nelson2001a,
 author = {Theodor Holm Nelson},
 title = {ZigZag (Tech briefing)},
 booktitle = {Proceedings of ACM conference on Hypertext and Hypermedia},
 year = 2001,
 pages = {261--262}
}
```

7.
```
@inproceedings{robertson2000a,
 author = {George Robertson},
 title = {From hierarchies to polyarchies: visualizing multiple relations
 booktitle = {Proceedings of ACM AVI 2000 working conference on Advanced
 year = 2000,
 pages = {13},
 location = {Palermo, Italy},
 doi = {http://doi.acm.org/10.1145/345513.345230}
}
```

8.
```
@inproceedings{robertson2002a,
 author = {George Robertson and Kim Cameron and Mary Czerwinski and Danie
 title = {Polyarchy visualization: visualizing multiple intersecting hier
 booktitle = {Proceedings of ACM CHI 2002 Conference on Human Factors in
 year = 2002,
 pages = {423--430}
}
```

9.
```
@article{robertson2002b,
  author = {George Robertson and Kim Cameron and Mary Czerwinski and Dani
  title = {Animated visualization of multiple intersecting hierarchies},
  journal = {Information Visualization},
  year = 2002,
  month = {March},
  volume = 1,
  number = 1,
  pages = {50--65},
  publisher = {Palgrave Macmillan Ltd.},
}
```

10. Michael Swaine, Dr. Dobb's Journal, December 1998. Excerpt at http://xanadu.com.au/mail/zzdev/msg02014.html
11. http://xanadu.com/
12. http://www.udanax.com/

13. http://xanadu.com/zigzag/
14. Theodor Holm Nelson. What's On My Mind. Invited talk at the first Wearable Computer Conference, Fairfax VA, May 12-13, 1998. http://www.xanadu.com.au/ted/zigzag/xybrap.html
15. http://xanadu.com/zigzag/tutorial/ZZwelcome.html
16. http://xanadu.com/zigzag/tutorial/zzConnex.html
17. http://xanadu.com/zigzag/tutorial/zzViews.html
18. http://xanadu.com/zigzag/tutorial/zzAxioms.html
19. http://xanadu.com/zigzag/tutorial/zzMultidims.html
20. http://xanadu.com/zigzag/tutorial/zzProperties.html
21. http://xanadu.com/zigzag/tutorial/zzParadoxes.html
22. ZZ Cell Programming. http://xanadu.com/zigzag/fw99/ZZcellProg.html
23. Ted Nelson. Confidential presentation of Ted Nelson's ZigZag(tm) and Ted Nelson's Dimensia(tm). http://www.xanadu.net/zigzag/manual/ March 24, 1994.
24. Gzz, formerly known as GZigZag. http://www.nongnu.org/gzz/
25. Gzz in a Nutshell. http://www.nongnu.org/gzz/nutshell.html
26. Tuomas Lukka & Katariina Ervasti. A Platform for Cybertext Experiments. http://www.nongnu.org/gzz/ct/ct.html
27. Tuomas Lukka. A Gentle Introduction to Ted Nelson's ZigZag Structure. http://www.nongnu.org/gzz/gi/gi.html

---

Email correspondence

(This section added June 2004)

After this document was pretty much finalized and published on the web, on 13 Jan 2004, I emailed Ted Nelson asking him to look at it. He wrote back, and offered answers to some of the "Open Issuse" in section 6. Below is my subsequent response, dated 22 Jan 2004, where I quote parts of his message.

```
> First, I am greatly honored by such a respectable analysis of the system.
> (Barbara Simons, past president of the ACM, has said she thinks ZigZag can
> be analyzed in graph-structure terms, and you are doing just that.)
>
> On a quick reading, I am hugely impressed and pleased that you've done
> this.

I'm glad to hear you seem to approve, on the whole.
I still don't feel like I fully appreciate ZigZag,
but what little I do understand, I find intriguing
enough that I think it's important to keep accurate,
detailed, and clear records of whatever ZigZag designs we can,
for posterity.  So getting approval from the originator
of ZigZag is quite desirable.


> My one question is why you use one-way arrows for your links, when all
> zzstructure links are inherently symmetrical?  I recognize that this is the
> graph-structure tradition, but some might find it misleading.

Indeed, I am following graph theory conventions with the one-way arrows.
The arrows are useful for differentiating posward from negward
directions.
I do not, however, mean that the links can only be traversed
```

```
in one direction, nor that the zzstructure must be implemented
with one-way pointers.  I'll probably add a note to my tutorial
to clarify this, once I find some more time to work on it.


> In any case, thank you ever so much for doing this work.
>
> ===A quick stab at your key questions--
>
>         - If a user has a large set of data stored in a zzstructure,
> involving
> many different dimensions, might the management of a large set of
> dimensions become a problem ?
>
>                 Ans.  Rather than write essays about this, I intend to put
> my answer into
> a product.  Also users may try their own solutions.
>
>         - What's the best way to visualize zzstructures on a 2D screen ?
> How can
> we make full use of available screen space ?
>
>                 Ans.  There are no determinate answers here.  But "full
> use" is really
> about responsiveness, not about static views.

Very well put.



> I want to get video-game
> performance out of this puppy.  We're currently developing in OpenGL.  (See
> the Windows prototype.)
>
>         - With traditional hierarchical trees, if a user wants to do an
> exhaustive
> visual search for something ... [Visual? /tn]...

By "visual search", I mean doing it manually, at the user's pace,
e.g. on linux this might correspond to doing "ls", "cd folder_name",
"ls", "cd ..", etc.  In this way, the user can exhaustively explore
a (smallish) subtree, and be sure that they encounter every file
exactly once.




> However, with
> zzstructures, as with even simple graphs, there doesn't seem to be any
> straight-forward traversal that will visit every node once and be easy to
> keep track of. Might nodes become lost or forgotten ... ?
>
>                 Ans.  Good question.  Search applets will require
> experimentation.
> Cataloguing can help (e.g. keeping track of which cells are coplanar.)
>
>         - How might we reveal all the dimensions along which a cell has
> connections...?
>                 Ans.  See Gzz, the beautiful implementation by Tuomas Lukka
> et al.  The
> latest version has a view that shows all a cell's connections.
```

Later, I received this message from Ted, dated 17 Mar 2004.

```
Hi Michael--

>I still don't feel like I fully appreciate ZigZag,

This is my impression also.

Here's the new one-liner:

"zzstructure is a new system of representation which generates
 a visible and traversible pseudospace, which has a variety
 of benefits ..."

I think it's very important that you try it interactively,
 still best done with the gzz version.
```

Later still, I received this message dated 16 May 2004.

```
Another point-- you asked how do searches--
>By "visual search", I mean doing it manually, at the user's pace,
>e.g. on linux this might correspond to doing "ls", "cd folder_name",
>"ls", "cd ..", etc.  In this way, the user can exhaustively explore
>a (smallish) subtree, and be sure that they encounter every file
>exactly once.
>> However, with
>> zzstructures, as with even simple graphs, there doesn't seem to be any
>> straight-forward traversal that will visit every node once and be easy to
>> keep track of. Might nodes become lost or forgotten ... ?

I didn't say this earlier-- the best thing is to create a View
 that finds all the stuff you want and shows it in ZigZag
 optimally and wonderfully :)
```

Substantially written (including Figures 1-17) during December 2003
Published on the web, with revisions (including Figure 18), during January 2004
Email correspondence section added June 2004