Space-Efficient Visualisation of Large Hierarchies

Quang Vinh Nguyen

A thesis submitted for the degree of
Doctor of Philosophy in Computing Sciences
at the
University of Technology, Sydney

Faculty of Information Technology
University of Technology, Sydney
Sydney, Australia

2005

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

Date	**August 05**
Author	**Quang Vinh Nguyen**
Title	**Space-Efficient Visualisation of Large Hierarchies**
Degree	**Doctor of Philosophy in Computing Sciences**

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate

Acknowledgements

I would like to take this opportunity to express my sincere gratitude to my supervisor, Dr. Mao Lin Huang, for his enormous academic guidance and financial support, encouragement, advice, and for giving me the opportunity to do a Ph.D at the Faculty of Information Technology at the University of Technology, Sydney. Mao Lin provides me not only a great academic guidance, but also a large degree of freedom in my research, for both of which I am very grateful.

I also would like to express my sincere thank to my co-supervisor, Prof. Chengqi Zhang, for his great help and useful advice.

I would like to thank Prof. Igor Hawryszkiewycz for giving me the opportunity to carry out one of my case study using the *LiveNet* system, which makes it much more valuable to my thesis. I have leant a lot from the experiment, as well as from the discussions with Igor. I also wish to acknowledge the great support of Dongbai Xue for his eager help when I worked with the *LiveNet* system.

Great thanks are due to A/Prof. Tom Hintz and all members of Computer Vision Research Group for their generous helps, supports, and the exchange of knowledge toward the completion of my Ph.D. I would acknowledge the effort of A/Prof. Tom Hintz for the English correction.

I would like to thank Prof. Peter Eades and his Information Visualisation group members, who have given me opportunities to share and exchange the knowledge during the group meetings at the University of Sydney, Australia.

I would also thank all of my colleagues, academics and technical staffs of Faculty of Information Technology, the Graduate School and the Vice-Chancellor's Conference Fund at the University of Technology, Sydney who directly or indirectly help me on the road toward the completion of my Ph.D, especially to Ryan Heise, Robert P. Biuk-Aghai, and Ai Zhong Lin.

The author would acknowledge the appreciation to the markers of Tree-Maps and Space-Tree software (University of Maryland's Human-Computer Interaction Lab). These two softwares are very useful for my research and experiment.

Last but not least, I would like to thank my parents, my grandma, and my sisters for their continuous encouragement and support. I would also thank my girlfriend, Kim Chi Nguyen, who has always stayed side by side to make this Ph.D possible.

To my beloved parents & my loved girlfriend

Table of Contents

List of Figures

List of Tables

List of Equations

Abstract

Relational information visualisation concerns viewing relational data, where the underlying data model is a graph. Hierarchical visualisation is one of hot topics in graph visualisation in which the data is organised in a hierarchical structure. As the amount of information, that we want to visualise, becomes larger and the relations become more complex, classical visualisation techniques and hierarchical drawing methods tend to be inadequate.

Traditional hierarchical visualisation algorithms are more concerned with the readability of the layouts. They usually do not consider the efficient utilisation of the geometrical plane for the drawings. Therefore, for most hierarchical layouts, a large portion of display space is wasted as background. The aim of this research is to investigate a space-efficient approach to handle the visualisation of large hierarchies in two-dimensional spaces.

This thesis introduces a new graph visualisation approach called *enclosure+ connection* for visualizing large hierarchies. This approach maximises the space utilisation by taking advantages of the traditional enclosure partitioning approach, while it retains the display of a traditional node-link diagram to hopefully provide users a direct perception of relational structures.

The main contribution of this thesis is layout and navigation algorithms for visualising large hierarchies. Two layout algorithms, the *space-optimised tree* and the *EncCon tree*, have been developed to achieve the space-efficient visualisation. Both algorithms use the enclosure concept to define layout of hierarchies, which ensure the efficient utilisation of display space. Two *focus+context* navigation and interaction methods have been proposed to cooperate with the visualization of large hierarchies. Several advanced computer graphics approaches, such as *graphic distortion* and *transparency*, are used for the development of these navigation methods.

Two case studies have been implemented to evaluate the layout algorithms and the associated navigation methods. The first case study is an application of a shared collaborative workspace which aims to provide users with a better assistance for visual manipulation and navigation of knowledge-based information. The second case study is a visual browser for navigating large-scale online product catalogues.

Although the case studies have provided some useful evaluation, formal usability studies would be required to justify fully the effectiveness of these layout and navigation methods. Although this task has not carried out in this research, the author has presented his usability study's plan as a future work.

Terminology

- **A graph *G = {V, E}*:** is defined as a pair (*V, E*), where *V* is a set of vertices, and *E* is a set of edges between the vertices $E = \{(u,v) \mid u, v \in V\}$.
- **A tree**: is a connected graph without a cycle.
- **A rooted tree:** consists of a tree *T(r)* and a distinguished vertex *r*. The vertex *r* is called the root of *T*. In other words, *T* can be viewed as a directed acyclic graph with all edges oriented away from the root. If (μ, ν) is a directed edge in *T*, we then say μ is the father of ν, or ν is a child of μ. If *T* contains vertex ν, then the sub-tree $T(\nu)$ rooted at ν is the sub-graph induced by all vertices on paths originating from ν.
- **A leaf vertex:** is a vertex with no children.
- **A node:** represents a vertex with its displaying properties.
- **Weight $w(\nu)$:** represents the weight of vertex ν.
- **Wedge $wg(\nu)$:** is defined by a vertex μ, line *l* goes through μ, and a clockwise angle $\alpha(\nu)$; where μ is the father of ν. Thus, we have $wg(\nu) = \{\mu, l, \alpha(\nu)\}$ (see Figure 2.3).
- **Local region $R(\nu)$:** is an area which contains the drawing of a sub-tree $T(\nu)$.
  - At the Space-Optimised Tree model: $R(\nu)$ is a polygon that is defined by the wedge $wg(\nu)$ and one (or more) cutting edges (boundaries of other local regions) that cross the line l in $wg(\nu)$ (see Figure 2.4).
  - At the EncCon model: $R(\nu)$ *is* a rectangle.
- **Layered visualisation *LV*:** consists of two graphical layers L_1 and L_2 of the information that are appeared in an overlapped manner in the visualisation, $LV = L_1 + L_2$ (see Figure 3.8). Each layer is the medium for the drawing of graph *G* or a sub-graph $G_l \in G$. At any time, a graph G_i drawn in L_1 is always a sub-graph of G_j drawn in L_2. Thus, we constantly have $G_i \in G_j \in G$.

Chapter 1 Introduction

For the last two decades, the amount of information has boomed to almost double every year. Thank to the fast growth of technology, the computational power of the current computers has made many graphic aids for users to amplify the understanding of information. Visualisation has become a crucial and expanding role in current cognitive systems. Interactive visualisation has been more and more popular in many domains, such as business, education, and science for analysis, study, and manipulation of data. From the study of Colin Ware (Ware 2004), visualisation displays provide the highest perception from the computer to human. In other words, more information can be absorbed from a visualisation than any other senses combined. The benefit of a good visualisation is sometimes said as "a picture is worth ten thousand words"[1].

1.1 Information Visualisation

Visualisation is formally defined as "the use of computer-supported, interactive, visual representations of data to amplify cognition" (Card, Mackinlay & Shneiderman 1999). In other words, *visualisation* is an aid for discovery, decision making, and explanation of information. Visualisation is divided into main streams including *scientific visualisation* and *information visualisation*. *Scientific visualisation* is seen primarily relate to, and represent visually a physical thing such as the human body, the earth, molecules, etc. *Scientific visualisation* techniques are usually rendered in three-dimension using powerful graphic workstations. On the other hand, *information visualisation* concerns with reducing the cognitive overheads of understanding complex information structures through the use of visual representations. The definition of *information visualisation* formally is "the use of computer-supported, interactive visual representations of abstract, non-physically based data to amplify cognition" (Card, Mackinlay & Shneiderman 1999). *Information visualisation* can be applied to numerous

[1] This quotation was simply made up from "Chinese Proverb" and appeared in the advertising trade journal then called Printers' Ink, March 10, 1927.

areas through the workplace and science, wherever information is being handled. In addition, *information visualisation* techniques do not often require powerful computers to run their applications. Although *information visualisation* is distinguished from *scientific visualisation*, these two fields do not "conflict", and are actually related. In some cases, the boundary of these two fields is blurred. More definition and description about *information visualisation* can also be found in (Chi 2002; Herman, Melancon & Marshall 2000; Spence 2001).

1.1.1 Origins in Information Visualisation

Information visualisation evolved from *scientific visualisation* in the early 1990's with the first use of the term *information visualisation* was Robertson, Card, and Mackinlay in 1989 (Robertson, Card & Mackinlay 1989). The first IEEE Visualisation conference was organised in 1990. This community was led by earth resource scientists, physicists and computer scientists in supercomputing, who were more concerned in *scientific visualisation*. The annual IEEE Visualisation conference now becomes the leading event in both scientific and information visualisation. The early proposed and implemented information visualisation techniques notably are *worlds within worlds* (Feiner & Beshers 1990), *tree-maps* (Johnson & Shneiderman 1991), and *information visualizer* (Card, Robertson & Mackinlay 1991), and others.

Since then, several refinements and new *information visualisation* methods have been proposed and implemented in several applications and communities. Along with IEEE Visualisation conference, there are several good conferences which directly or indirectly concern with information visualisation. Notably, they are the *Graph Drawing Symposia*, the *CHI'XX, UIST'XX* conferences, and the *Information Visualization IV'XX*, etc. At the time of writing, the information visualisation journal, published by *Palgrave Macmillan*[2], has started more than three years ago. This event indicates the maturity of research in *information visualisation* as well as the public concern for this topic.

[2] http://www.palgrave-journals.com/ivs/

1.1.2 Human Visual Perception

Although a good visualisation can benefit users for rapid interpretation of large quantity of information, a bad visualisation can also cause the distortion or obscuration of the data which make much harder for viewers to understand or compare. Consequently, in addition to several good visualisation techniques, there are also a large numbers of poor visualisations which are responsible for misunderstanding, misinterpretation and misjudgement. Figure 1.1 shows an example of one of the best information visualisation techniques in which the disastrous result of Napoleon's failed Russian campaign in year 1812 is well illustrated. Figure 1.2 also shows an example of a bad information visualisation technique which purports to show the mandated fuel economy standards set by the US department of transportation. Although this figure tries to show an increase in mileage 53%, the magnitude of increase shown in the graph is 783%. More typical good and bad examples of data visualisation are critiqued by Friendly (Friendly 2000). This raises a critical question "what is the best way to transform data into visual images or diagrams so that people can easily understand and interact with them".

Human visual perception plays an essential role as a guideline in designing a good information visualisation system. Most information displays visualise information on computer monitors which are only a single rectangular planar surface. Thus, the perception of users might vary and bear on a number of display problems due to the mismatch of the eye and the display devices. These common problems are possibly from visible light, brightness, colour, contrast, size of the display device, resolution, etc and most importantly the quality of the lens system of the human eye.

Colin Ware discussed the visual efficiency of a display screen for effective perception of a human eye (Ware 2000, 2004). In detail, he aimed to find out the optimal screen size that provides the best match of screen pixels to brain pixels. This study is very important as a design guideline for future large-scale information visualisation research, which ensures the maximum information display and retains a good aspect for perception. The research shows that more information can be displayed on very high-resolution and large screen, but it does not necessarily provide very much more information into the brain. This is because the conventional monitor covers only 5-10% of visual field in the normal condition, but it uses as much as 50% of brain pixels (Wilkins 1995). The study also shows that the uniquely stimulated brain pixels peak at the width of a normal monitor view, and it is effective (but not critical) to increase the number of pixels for the normal desktop to reach the limit of the brain pixels.

As a result of the above studies, the deployment of a large and high resolution display does not guarantee significant improvement of visualisation compared to a standard display because of the constraint limits of our brain pixels. Therefore, researching a new optimal and efficient technique for visualising large datasets is more effective and efficient than expensive large display devices.

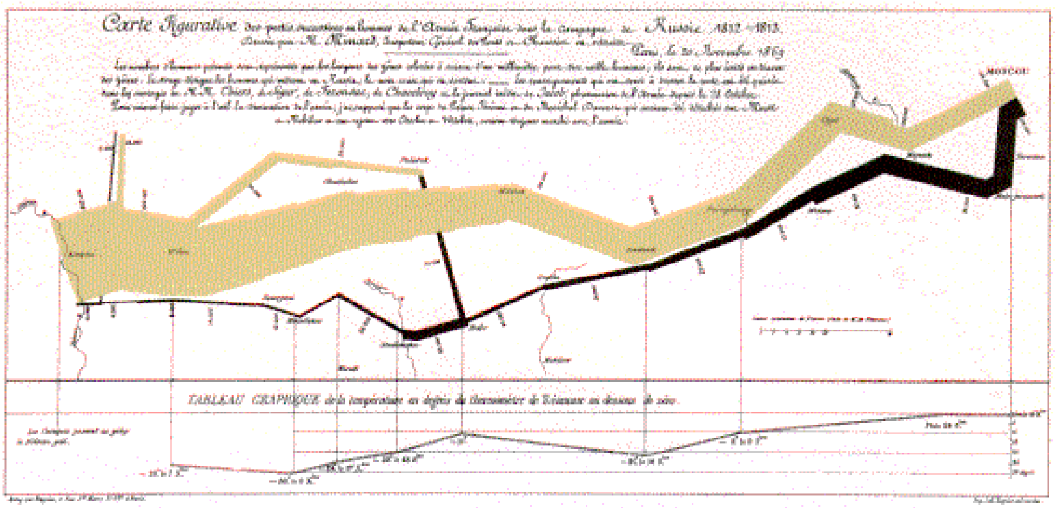


Figure 1.1. An example of good information visualisation.
Mathematica graphic of Minard's depiction of the fate of Napoleon's army.
Source from: (Shaw & Tigg 1993).

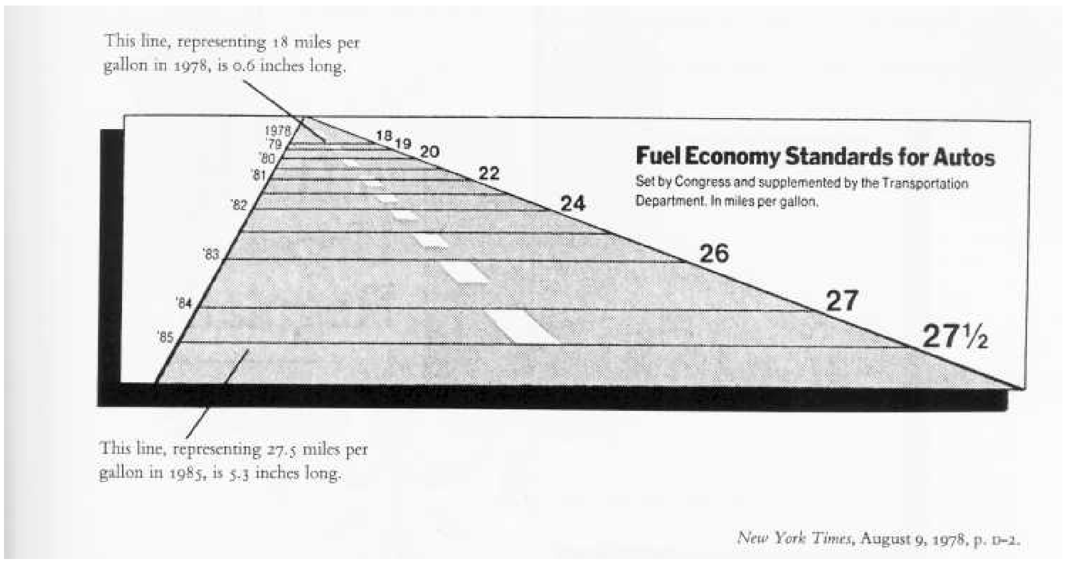


Figure 1.2. An example of bad information visualisation.
The lie factor? Source from: (Friendly 2000).

1.2 Graphs in Information Visualisation

Graph visualisation or graph drawing is the visualisation of relational information in which the structures of data is fundamentally modelled as graphs. Formally, it is defined as the "problem of constructing geometric representations of graphs, networks, and related combinatorial structures" (Di Battista et al. 1999). In the *graph visualisation*, nodes are represented as data items and edges are represented as relations. In several graph drawing applications, nodes are simply presented as small rectangles with or without labels, and edges are presented as straight or curved lines. *Graph visualisation* has a wide range of applications such as *data flow diagrams*, *entity-relationship diagrams*, *organisation chats*, *state-transition diagrams, networks*, and many more.

Graph drawing research is divided into three directions including discrete mathematics, algorithmic and human-computer interaction (Di Battista et al. 1999). Discrete mathematics concerns the theories of graph topology, graph geometry and graph order. Research in graph algorithms concentrates on graph algorithms, data structures, and computational geometry. Human-computer interaction is more about the aspects of human perception and interaction, including visual languages, graphical user interfaces, software visualisation, and so forth.

A large number of graph drawing techniques use node-link diagrams to present the data and relationships among the items. These techniques are more concerned with aesthetics rules in the design of layouts. The aesthetical niceness' rules of a graph layout are commonly: *minimising edge crossings*, *displaying symmetry of graph structures*, *minimising bends in edges*, and *keeping multi-edge paths as straight as possible*. However, there is still little research on the empirical validation of the above layout aesthetics, or the perception and cognition in graph aesthetics (Ware et al. 2002).

1.3 Hierarchical Visualisation

Hierarchical visualisation is a sub-section of the graph visualisation and it is one of the *hot topics* in information visualisation. In this type of the visualisation, the data is organised in a hierarchical structure, and in graph theory, we usually call these structures as hierarchical trees. The importance of hierarchical visualisation is highlighted by the nature of a large quantity of hierarchical data in the real world, such as *structures of classification systems*, *taxonomies of animals*, *product catalogues*, *file systems*, *organization chats*, etc. Chaomei Chen (1999) stated that:

> *Hierarchies are one of the most commonly used structures... Hierarchical structures not only play significant roles in their own right, but also provide a means of representing a complex structure in a simplified form.*

The visualisation of hierarchical data improves not only the understanding of data and the relationship between data items, but also assists in manipulating information. Research in hierarchical visualisation can be roughly classified into two main streams: the *connection* approach and the *enclosure* approach. They are both effective approaches for the visualisation of hierarchies and which one we should use depending primarily on the properties of the data in a particular application domain.

- The *connection* approach displays the relationships of information explicitly by drawing a set of graphical edges that gives users an immediate perception of the relationships (see an example at Figure 1.3). However, connection visualisation is not often efficient in term of utilising display space. Typical techniques in connection approach are classical *hierarchical view* (Reingold & Tilford 1981), *h-tree layout* (Shiloach 1976), *radial view* (Eades 1992), *balloon view* (Melancon & Herman 1998), *disk tree* (Chi et al. 1998), *NicheWorks* (Wills 1999), *rings* (Teoh & Ma 2002), *Narcissus* (Hendley et al. 1995), *hyperbolic browser* (Lamping & Rao 1996; Munzner 1997), *cone-tree* (Robertson, Mackinlay & Card 1991), *botanical visualisation* (Kleiberg, van de Wetering & van Wijk 2001), etc.

- The *enclosure* approach, typically *tree-maps* (Johnson & Shneiderman 1991), *cushion tree-maps* (van Wijk & van de Wetering 1999), *squarified tree-maps* (Bruls, Huizing & van Wijk 2000), *Venn diagram* (Herman, Melancon & Marshall 2000), etc., is a better solution in term of optimising the use of display space (see an example at Figure 1.11). However, these techniques do not show explicitly the relational structures of information. This costs extra cognitive effort of viewers to understand the relational structures that are presented implicitly in the enclosure manner.

Although many techniques have been proposed and implemented, the size of the datasets is still a key issue in graph and hierarchical visualisation. Large graphs and trees can decrease significantly the performance of a visualisation technique which normally performs well on small or medium datasets. In addition, the issue of "view-ability" also arises because it will be impossible to discern between nodes and edges (Herman, Melancon & Marshall 2000). As a result, large graph visualisation, or large hierarchical visualisation in specific, has become one of the hot topics in information visualisation.

The main concerns of information visualisation techniques are not only the geometrical positioning of the hierarchy but also the navigation and interaction. This is because no layout algorithms can work alone to overcome the problem of visualising and navigating large datasets using limited display spaces. Therefore, associated techniques for interactive navigation are becoming the essence in the design of visualisation of large size information spaces.

We now review and discuss current hierarchical visualisation techniques.

1.3.1 Hierarchical Layout Techniques

Research in hierarchical visualisation in two-dimensional (2D) space can be roughly classified into two main streams the *enclosure* and the *connection*. They are both effective approaches for the visualisation of hierarchies and which one we should use depending primarily on the properties of the data in a particular application domain. *Connection* approach displays the relationships of information explicitly by drawing a set of graphical edges in the diagram that gives users an immediate perception of the relationships. *Enclosure* approach is a better solution in term of optimising the use of display space. However, techniques in this approach do not show explicitly the relational structures of information. This can cost extra cognitive effort of viewers to understand the relational structures that are presented implicitly in the enclosure manner.

Three-dimensional (3D) hierarchical layout techniques are another approach that uses an extra dimension to achieve the display of more information on the screen. These techniques also aim to provide the "natural look" of the information. Three-dimensional layout algorithms can also use enclosure to partition the layout or use node-link diagrams in their displays. However, 3D techniques might suffer from viewing problems due to occlusion caused by the three-dimensional metaphor. In addition, the computational time is also high to run those three-dimensional techniques. Some typical layout techniques are further discussed below.

1.3.1.1 Connection Approach

The *connection* approach is a natural way to draw a graph/tree structure in a node-link diagram. A set of visible graphical edges are drawn in the diagram to link nodes from the parents to their children. The nodes present the data while these edges are used to present relationships among data items. Considerable research in this area has been carried out including *hierarchical view* (Reingold & Tilford 1981), *h-tree layout* (Shiloach 1976), *radial view* (Eades 1992), *balloon view* (Melancon & Herman 1998), *disk tree* (Chi et al. 1998), *NicheWorks* (Wills 1999), *rings* (Teoh & Ma 2002), *Narcissus* (Hendley et al. 1995), *hyperbolic browser* (Lamping & Rao 1996; Munzner 1997), *cone-tree* (Robertson, Mackinlay & Card 1991), *botanical visualisation* (Kleiberg, van de Wetering & van Wijk 2001), etc.

The advantage of using a node-link diagram to present hierarchical structures is that the human can directly see the relationships that are drawn as a set of graphical edges appearing in the diagram. This makes it easier for the process of human perception to understand the relational structures of the information. Typical connection techniques for visualising large hierarchies are discussed below.

The classical hierarchical view - The technique is based on the algorithms developed by (Reingold & Tilford 1981). The *classical hierarchical view* uses a modular approach to the positioning of nodes where child nodes are positioned “below” their common ancestor. The algorithm calculates independently the relative positions of sub-trees and then joins them in a larger tree by placing these sub-trees as close together as possible. The parent node is located at the centre above. The layout can vary as top-down, left-to-right tree and grid-like positioning. The algorithm is simple, fast and predictable. The *classical hierarchical view* is simple and widely used in many applications. The layout, however, tends to expand over the display area from one dimension. In addition, the close positioning of nodes might cause problem in labelling. As a result, this technique is not adequate for visualising large size hierarchies. Figure 1.3 shows an example of a visualisation of a small datasets using *classical hierarchical view*.

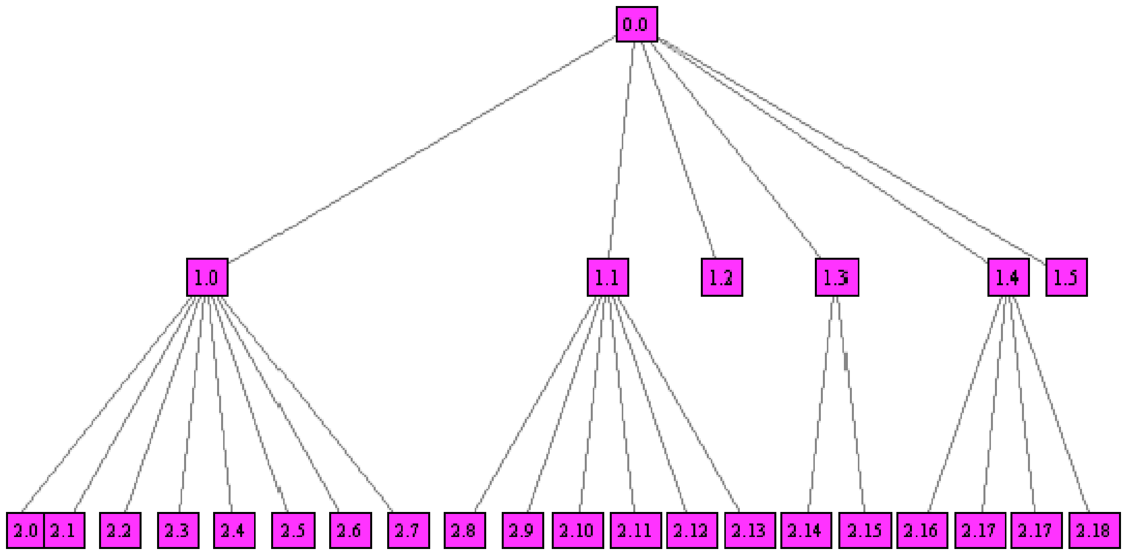


Figure 1.3. A visualisation of classical hierarchical layout.
Adapted from: (Reingold & Tilford 1981).

The radial view - This visualisation is based on an algorithm described in (Eades 1992). The algorithm recursively positions children of a sub-tree into circular wedges. The angles of these wedges are proportional to the number of leaves of the sub-tree. In order to avoid an angle assigned to a node being too large, *radial view* layout forces all wedges remain convex. The algorithm is simple, predictable and it behaves well in general. This technique, however, is also not optimal in the use of available space (see Figure 1.4a). This problem is overcome by simply dropping the convexity constrain property. In Figure 1.4b, we see that the space is used more efficiently. The new algorithm, however, also leads to intersections when applied to big trees.

(Jankun-Kelly & Ma 2003) presented a variation from the original *radial view* (Eades 1992), called *MoireGraphs*, for visualising graphs. This technique combines a radial graph layout with a number of interaction techniques to achieve its *focus+context* visualisation. This technique produces reasonably nice layout and it is capable of visualising hundreds of nodes, as claimed by the authors. Similarly to original *radial view*, *MoireGraphs* does not concern about space utilisation which make it difficult to visualising larger graphs or hierarchies with thousands of elements.

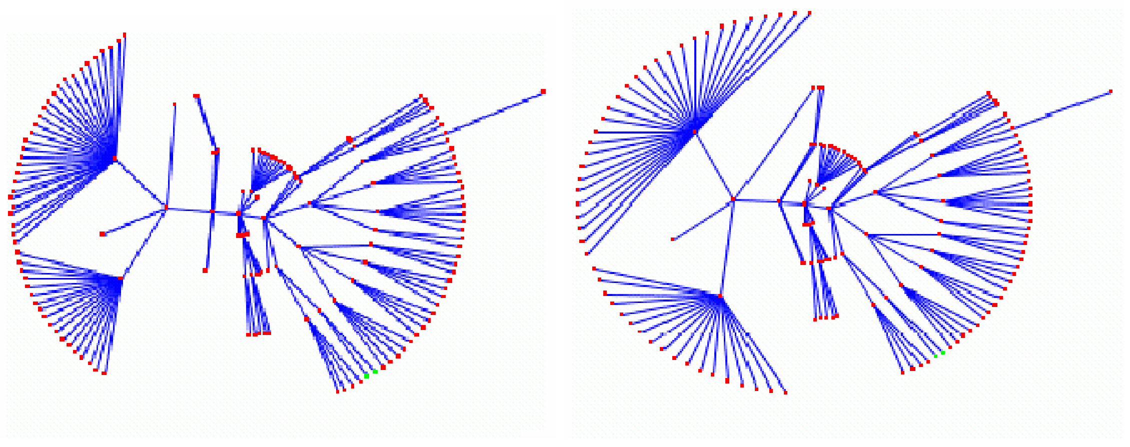

(a) (b)

Figure 1.4. A visualisation of radial view.

(a) Radial view with convexity check and (b) radial view without convexity check and with statistical modification. Source from: (Herman et al. 1999).

The balloon view (or circular view) - This layout is formed where siblings of sub-trees are included in circles attached to the parent node. In specific, the algorithm places every node at the centre of a circle, and each sub-tree rooted at the node is drawn into a smaller radii circle and is positioned on the circumference of the large circle (see Figure 1.5). The *balloon view* can also be obtained by projecting a *cone tree* (Robertson, Mackinlay & Card 1991) onto a plane. The algorithm behaves well on balanced trees. In this algorithm, notation of a root is temporarily based on user interest. This means that the user can interactively select an arbitrary node to be a root, and the full tree is reorganised based on this change. This property is an advantage of the *balloon view* in many applications. Like the *radial view* (Eades 1992), this layout wastes a large portion of the display space.

(Teoh & Ma 2002) presented an optimised circular view called *RINGS* which aimed to show more contextual information without loosing the clarity of the area in focus. The *RINGS*'s layout was based on a new ringed circular tree layout algorithm in which all nodes and their children are placed in circles. In summary, this technique places nodes as equal-sized circles in concentric rings around the centre of the parent circle. These circles can be either the large circles (for node with more children) or small circles (for node with fewer children) which are placed at the respectively outermost ring or at the inner ring. This technique also employs several visual cues such as colours and transparency to enhance the perception of graphs or trees. The authors

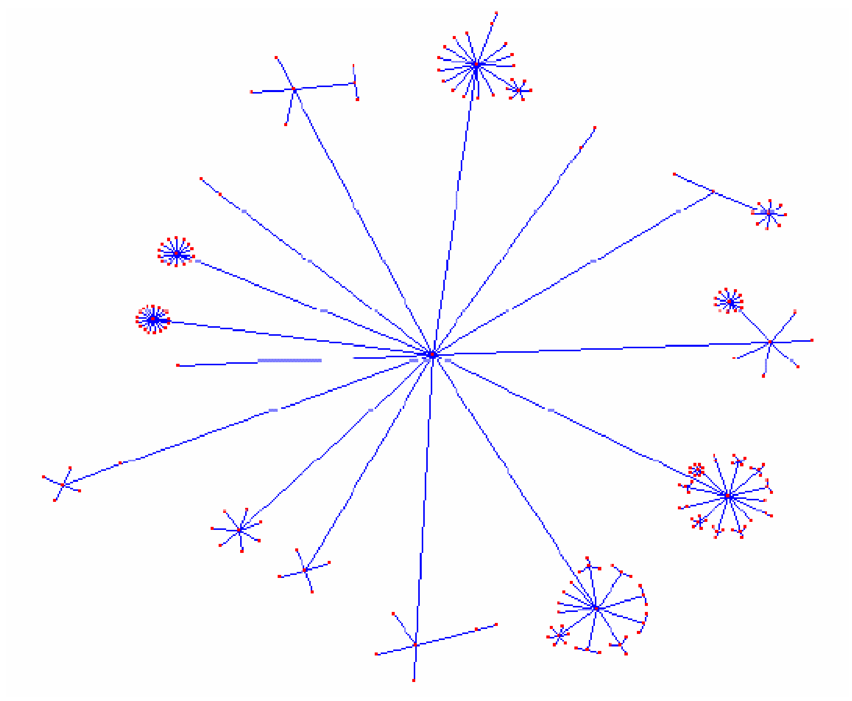


Figure 1.5. A visualisation of balloon view.

Source from: (Melancon & Herman 1998).

claimed that the *RINGS* is very efficient in term of utilising the limited display space, and indeed it is more optimal than the original circular view. However, this technique does not totally optimise the display space because of the different manners of circular layout and the rectangular display window.

Disk-tree – this technique also uses a circular layout to visualise the hierarchy and it is mainly used to visualise the evolution of web ecology in term of time tube visualisations (Chi 1999; Chi et al. 1998). In its layout, each successive circle indicates a level in a tree. Firstly, the algorithm calculates the number of leaf nodes and then the amount of angular space of each leaf node. Secondly, the algorithm traverses the hierarchy using breath-first traversal, and it calculates the angular space for each node based on the number of leaf nodes rooted at the current node (see Figure 1.6). *Disk-tree* is a good two-dimensional visualisation technique with the advantage that the entire structure of the information is visualised compactly and the layout is pleasant to view. Although *disk-tree* is a good technique for visualising large tree, the algorithm is not optimal in terms of using display space, especially the areas near the border. In addition, this method does not have a good navigation algorithm in order to interactively browse through the hierarchies.

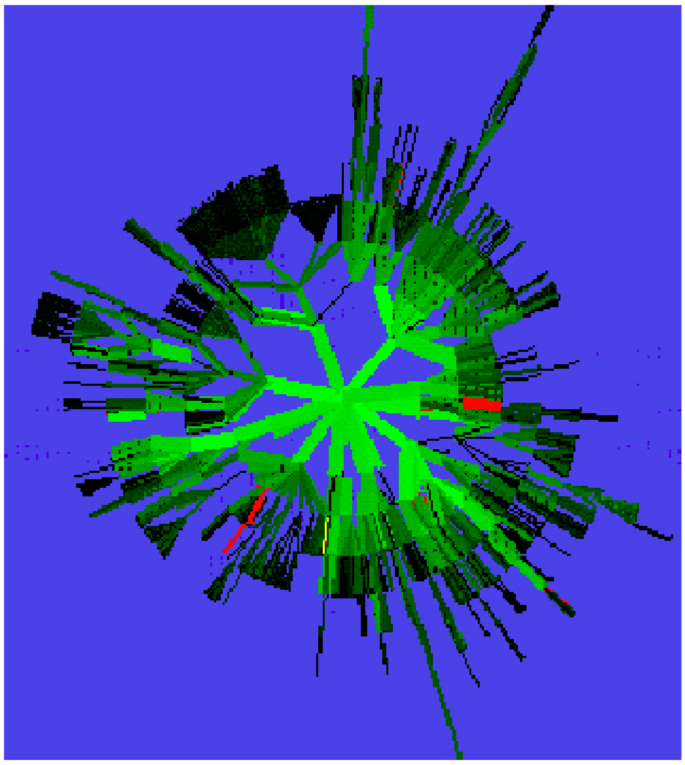


Figure 1.6. A visualisation of disk-tree.

Source from: (Chi 1999).

The hyperbolic browser - Hyperbolic layout was initially developed by Lamping and Rao (1996). This technique can be described informally as:

- The root node is initially placed at the centre of a circular display.
- All subordinate nodes of the root are arranged around it, and similarly to their own subordinate nodes, and so on.
- When the root is moved outward from the centre, the entire hierarchy is distorted that it fits within the circular display area.

Technically, this technique constructs the hierarchy in *hyperbolic geometry* and then maps that structure into ordinary *Euclidean plane*. The algorithm produces nice tree visualisation inside a disc and it provides an excellent focus+context navigation technique. As a result, it is quite capable for visualising large hierarchies (see Figure 1.3). *Hyperbolic browser* can be implemented in either a two-dimensional plane (Lamping & Rao 1996) or a three-dimensional space (Munzner 1997). The original two-dimensional *hyperbolic browser* uses *Poincare* model, while the three-dimensional

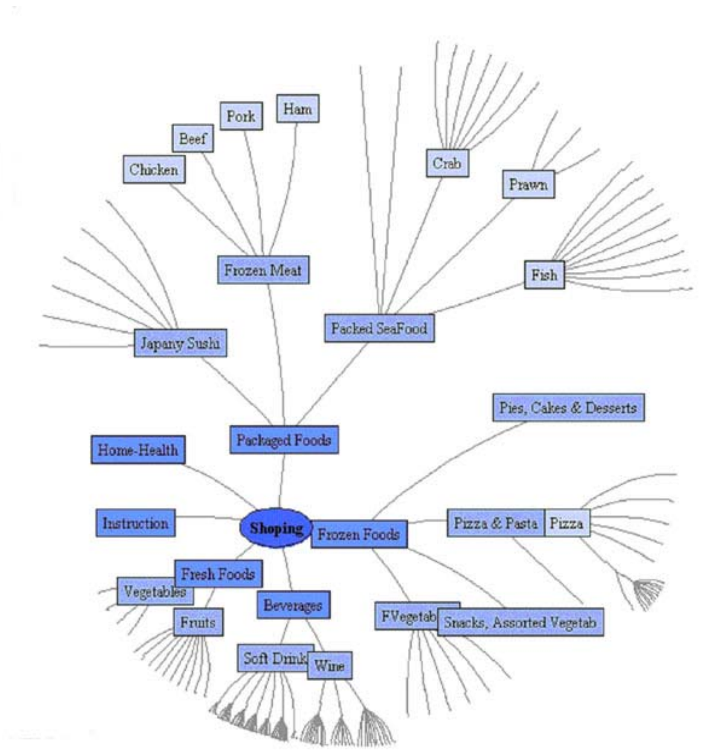


Figure 1.7. A Visualisation of hyperbolic browser.
Adapted from: (Lamping & Rao 1996).

hyperbolic browser uses *Klein* model in its implementation. There are several commercial implementations, also called *star tree*, are available at Inxight-Xerox. Although *hyperbolic browser* is a good layout technique, its algorithm is complex and hard to understand due to its different geometrical background.

Internet mapping - Cheswick et al. (2002) presented a nice technique for Internet and network mapping. The technique yields very large tree-like structures (100,000 nodes and more) by using a node-spring approach and a number of heuristics. The authors apply a force-directed method to lay out the graph. The standard model uses spring attraction to pull connected nodes together and it also employs 'electrical repulsion' to repel these nodes. The algorithm is relatively simple and produces a nice graph layout. However, this layout technique is rather slow: A typical run requires 20 CPU hours on a 400 MHz Pentium (Cheswick & Burch 2002). Figure 1.8 shows an example of the *internet mapping* visualisation with a dataset of nearly 100,000 nodes.

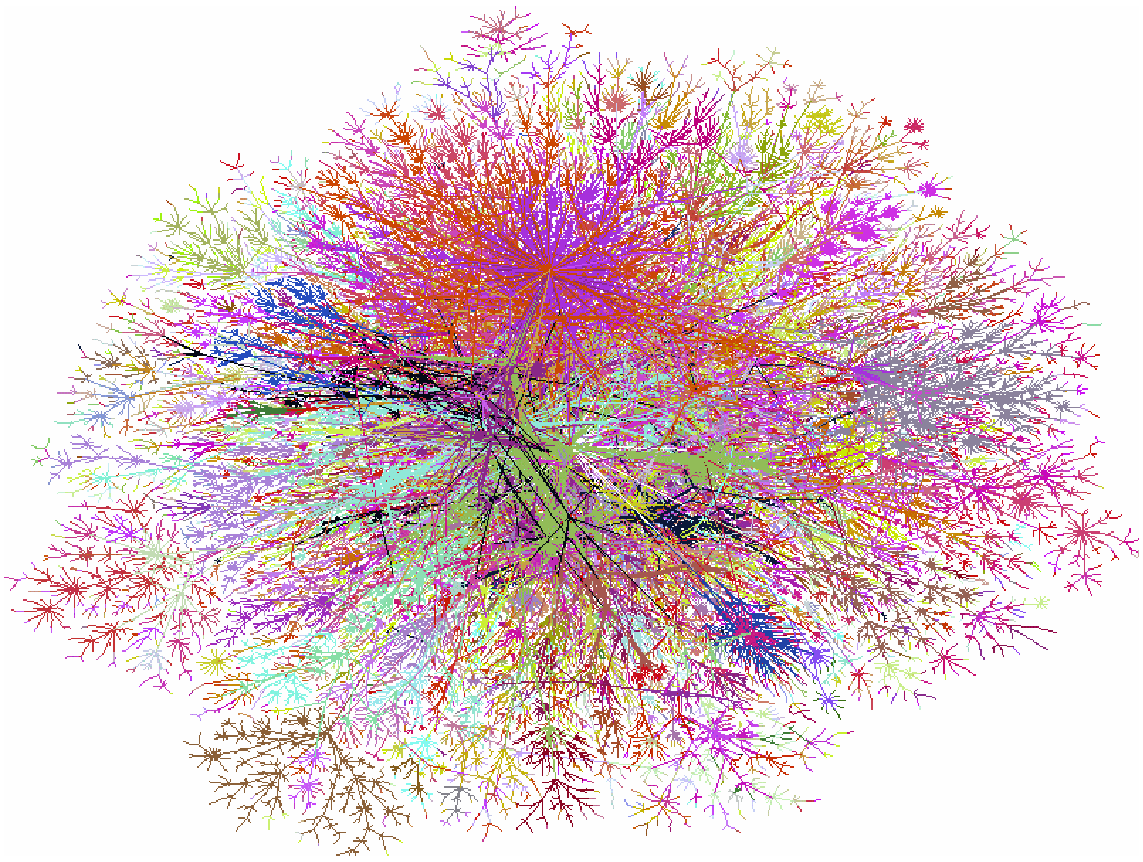


Figure 1.8. A visualisation of internet mapping.

Source: (Cheswick & Burch 2002).

NicheWorks - *NicheWorks* (Wills 1999) is a tool for visualising very large graphs. This system was originally designed to examine large telecommunication networks and it has been generalised to different areas including relationships between functions and files in a large software development effort, website analysis and correlation analysis in large databases. In the *NicheWorks* system, a user can examine a variety of node and edge attributes in conjunction with their connectivity information. The layout of this system includes two types of algorithms: initial layout and incremental algorithms. The initial layout is fast which is capable of visualising up to a million nodes in a few minutes and it includes circular layout, hexagonal grid and tree layout. The user may then use incremental algorithms to improve the graph layout. Finally, the system places all the components close to each other with the largest at the centre. This technique can show or hide parts of a graph by using interactive manipulation of views of node and edge attributes. Figure 1.9 shows a visualisation example of the *NicheWorks*.

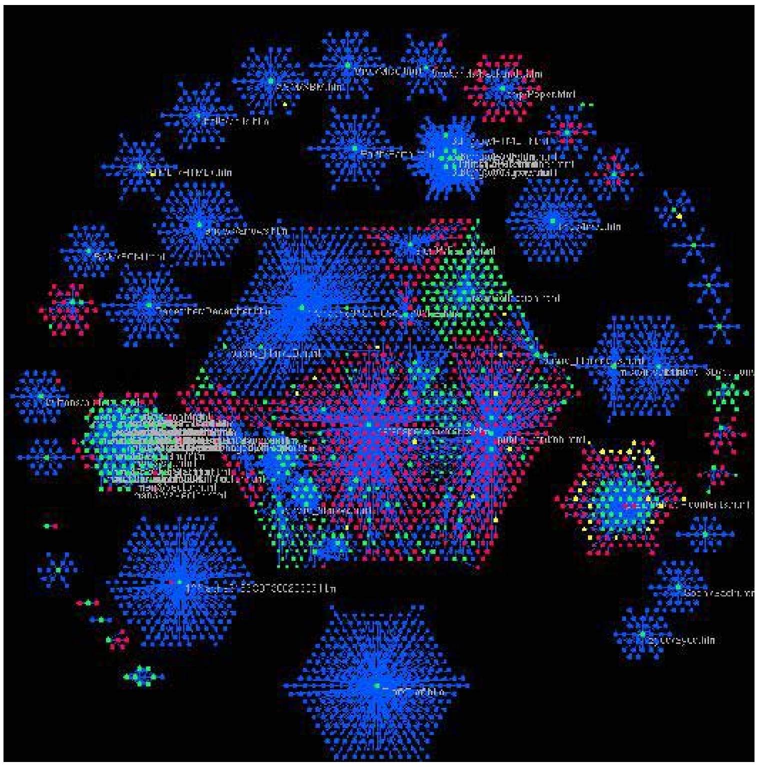


Figure 1.9. A visualisation of NicheWorks.

Source from: (Wills 1999).

1.3.1.2 Enclosure Approach

Another way to visualise hierarchical data in two-dimension is to use enclosure concept. Unlike the *connection* approach, the *enclosure* is the method of using enclosure to represent the tree structures. This partitioning ensures that all nodes and their sub-hierarchies locate inside their father's display region. Figure 1.10 shows a typical example of connection partitioning that each node is mapped to a rectangular area, then that area is subdivided in horizontal or vertical direction to show the relative size of the children of the node. There is also some extensive research in this direction. Typical examples of this type of visualisation techniques are *tree-maps* (Johnson & Shneiderman 1991), *cushion tree-maps* (van Wijk & van de Wetering 1999), *squarified tree-maps* (Bruls, Huizing & van Wijk 2000), *sunburst* (Stasko & Zhang 2000), *inclusion tree* (Eades, Lin & Lin 1993), *expand-ahead* (McGuffin, Davison & Balakrishnan 2004) and *Venn diagram* (Herman, Melancon & Marshall 2000).

The *enclosure* approach is a better solution in terms of optimising the use of display space. However, techniques in this approach do not show explicitly the relational structures of information. This costs extra cognitive effort of viewers to understand the relational structures that are presented implicitly in the enclosure manner. Two typical enclosure techniques, *tree-maps* and *sunburst*, for visualising large hierarchies are further discussed below.

Tree-maps – Unlike the *connection* approach, *tree-maps* (Johnson & Shneiderman 1991) represents hierarchical structures using enclosure for the partition of the display space to visualise the structural information. The drawing algorithm draws a node itself within its rectangular area with its display properties. The bound of the root is the entirely rectangular display area. Then the node sets a new bound within its rectangular bound and drawing properties for all its children. This process is repeated until all the leaves are reached. Figure 1.10 shows an example of the *tree-maps* visualisation that each node is mapped to a rectangular area, then that area is subdivided in horizontal or vertical directions to show the relative size of the children of the node. The process is recursively applied to the children nodes with the subdivisions on the X- or Y-axis.

There are two major variations of the *tree-maps* system namely *cushion tree-maps* (van Wijk & van de Wetering 1999) and *squarified tree-maps* (Bruls, Huizing & van Wijk 2000). The layout algorithm of *cushion tree-maps* is similar to the standard *tree-maps*, but this technique uses intuitive shading to provide insight in the hierarchical structure. Thus, this 3D shading is more effective and clearer than the original tree-maps. On the other hand, the *squarified tree-maps* modifies the original horizontal and vertical partitioning layout algorithms which ensures all the partitioned regions are roughly squared. This approach generally also provides better layout compared to the standard *tree-maps*.

The *tree-maps* and its variations, as were claimed by (Johnson & Shneiderman 1991), optimises all available space for display information. These techniques can provide an overview of an entire hierarchy, as well as the detail view of the focus sub-hierarchy. As a result, it makes the navigation of large hierarchies much easier especially when the quantitative variable is concerned. However, the lack of edges linking between nodes might prevent viewers from understanding the relationships of information because these relationships are not directly shown. Thus, it is much harder for the viewers to perceive and understand the structure of the relational hierarchies.

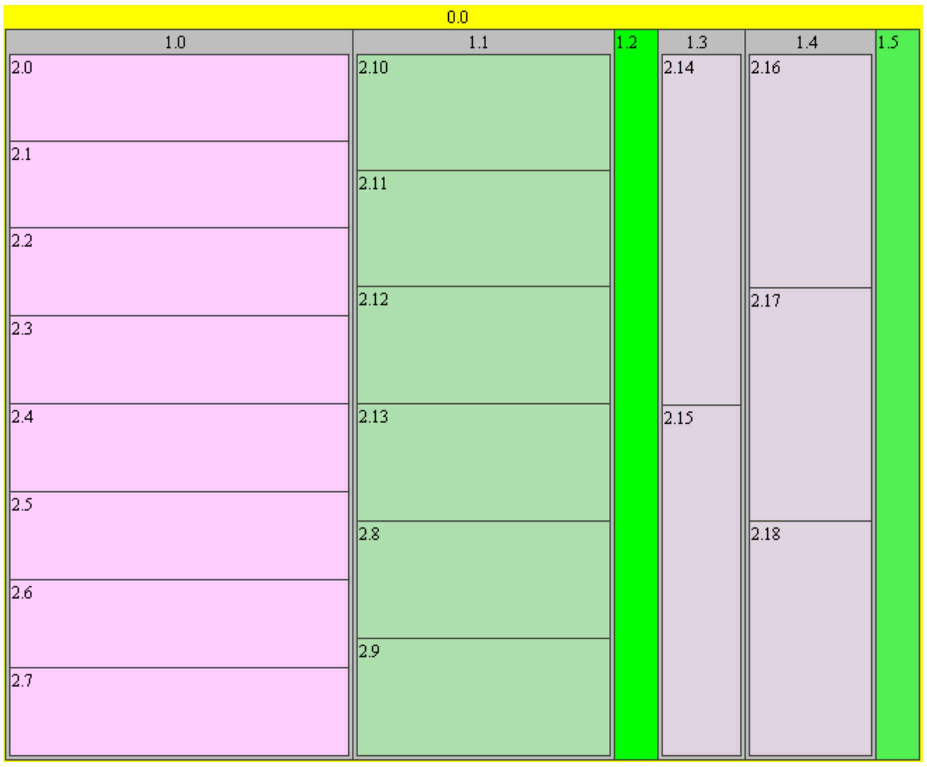


Figure 1.10. A visualisation of tree-maps.
Adapted from: (Johnson & Shneiderman 1991).

Sunburst – As described above, *tree-maps* partitions a hierarchy rectangularly so that the display region of a node is a rectangle. The *sunburst* technique (Stasko & Zhang 2000) also uses enclosure to partition the hierarchy, but in a circular or radial manner (see Figure 1.11). The layout algorithm is based on the other radial space-filling techniques from the circular visual designs (Chuah 1998) and information slices (Andrews & Heidegger 1998). These radial space-filling techniques were proposed as an alternative approach since some of the viewers might have better perception with a radial layout than a rectangular layout. However, the radial techniques are not fully optimised to the display space because of the different manner of a rectangular display space and the circular geometry. Similar to the *tree-maps*, the *sunburst* does not use edges to show the relations among nodes that increases the cognition of the user to perceive the relational information.

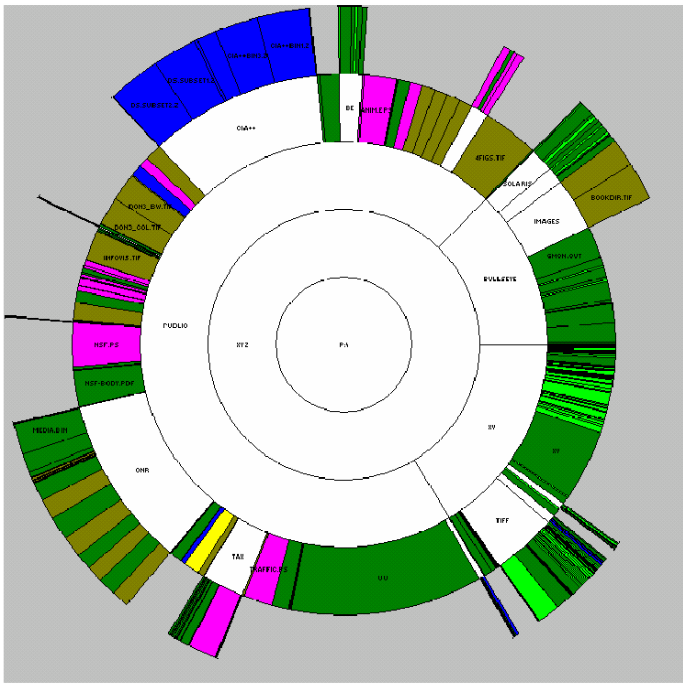


Figure 1.11. A visualisation of sunburst.
Source from: (Stasko & Zhang 2000).

1.3.1.3 Three-Dimensional Layouts

With the growth of technology, personal computers (PCs) have become much more powerful with several enhanced three-dimensional (3D) devices. These technological advances make three-dimensional information visualisation techniques feasible on PCs. Within the last decade, several 3D techniques have been invented for both research and commercial purposes. Thus, three-dimensional information visualisation has become more and more common as an alternative approach for visualising large information spaces such as networks, financial data, Internet, etc.

There are several approaches to construct a three-dimensional information visualisation. The simplest way is to generalise from two-dimensional (2D) layout algorithms to create a 3D visualisation. Similar to 2D layout techniques, 3D visualisations can also be classified as *connection* and *enclosure* approaches although the boundary is not as clear as from 2D. The *connection* approach uses node-link diagrams to indicate the relations among data items, such as *cone tree* (Robertson, Mackinlay & Card 1991), *botanical visualisation* (Kleiberg, van de Wetering & van Wijk 2001), *information landscape* (Andrews 1995), *3D narcissus* (Hendley et al. 1995), *3D hyperbolic browser* (Munzner 1997), *SeeNet* (Cox, Eick & He 1996), *spanning trees* (Herman, Melancon & Marshall 2000), etc. Three-dimensional *enclosure* techniques use enclosure to define the layout in two approaches. In the first approach, the layout is partitioned in a plane and then extruded to three-dimensional space. Typical techniques in this approach are *information pyramids* (Andrews, Wolte & Pichler 1997), *information garden* (Crossley et al. 1997), etc. In the second approach, the layout is partitioned directly in 3D as nested boxes in which the sub-hierarchies are always located inside their parent space. Typical techniques in this approach are *information cube* (Rekimoto & Green 1993), *GeoGraph3D* (Ghezzi 1997), and others. There are also some variations of nested design that only adopt partly the nesting metaphor, such as *WebBook and Web Forager* (Card, Robertson & York 1996).

The main objective of three-dimensional layouts is to use an extra dimension to achieve the display of more data on the screen as well as to provide the “natural look” of the information. 3D information visualisation shows a promising future with great support and effort from visualisation researchers. However, there are also several limitations of this approach. Firstly, creating a 3D visualisation is considerably more difficult than creating a 2D system. Secondly, three-dimensional techniques might suffer from viewing problems due to occlusion caused by the three-dimensional metaphor. In addition, there is a large amount of computational time is required to run these techniques. (Ware 2004) also shown that although 3D information visualisation was better in some tasks, a 2D solution was generally a better choice.

This thesis primarily concerns about two-dimensional hierarchical information visualisation. Within the scope, only three typical 3D information visualisation techniques are reviewed and discussed in detail.

Cone tree (or cam tree) – Since firstly introduced in 1991 by (Robertson, Mackinlay & Card), *cone tree* has become one of the best known three-dimensional graph/tree layout techniques. The layout is mathematically quite simple in which nodes are placed at the apex of a cone with its children placed evenly along its base (see Figure 1.12). *Cone tree* provides a nice technique for interaction and navigation that some labels of the nodes are transparent. The user can also pick any node and rotate the cone tree so that the chosen node is brought to front. This original idea of cone tree has been revisited and refined by Carriere and Kazman (1995). *Cone tree* is a good technique for visualising balance tree. The method, however, has a few disadvantages that large three-dimensional trees become too cluttered, the execution requires large computational time and it is incapable of displaying multiple hierarchical data structures.

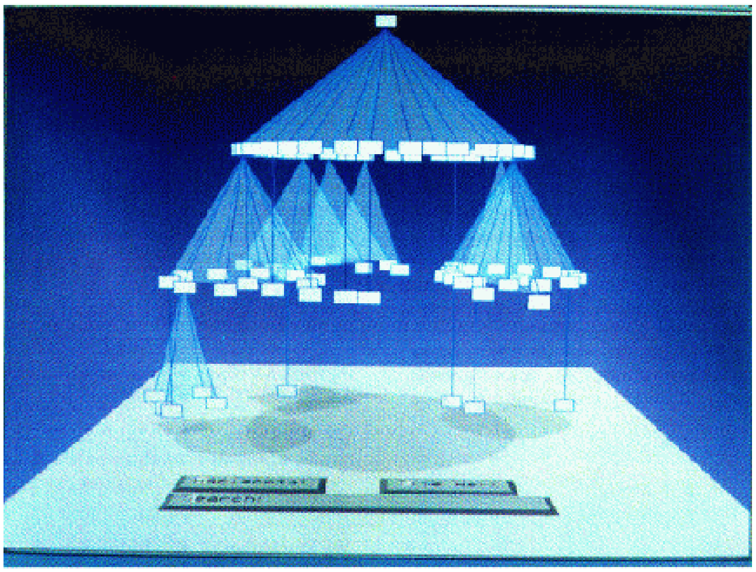
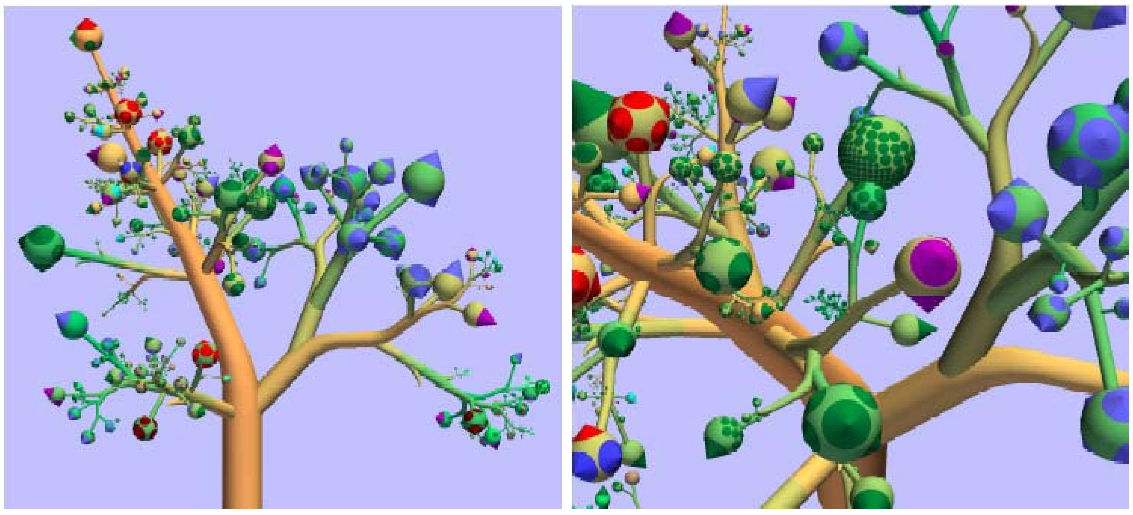


Figure 1.12. A visualisation of cone tree.

Source from: (Robertson, Mackinlay & Card 1991).

Botanical visualisation – *Botanical visualisation* is a new method for the visualisation of huge hierarchical data structures. In general, this technique uses the botanical-tree metaphor to visualise the huge hierarchy in three-dimensional space (Kleiberg, van de Wetering & van Wijk 2001). The visualisation provides beautiful layouts for the hierarchies where information is arranged as branches and leaves on a botanical tree (see Figure 1.13a). However, this technique only supports its navigation through the zooming technique in which the context is lost during the navigation (see Figure 1.13b). The large rendering time is also a limitation of this technique.



(a) (b)

Figure 1.13. A visualisation of botanical visualisation.

Source from: (Kleiberg, van de Wetering & van Wijk 2001).

Information cube – this visualisation can be considered as the generalisation of *tree-maps* (Johnson & Shneiderman 1991) in three-dimensional space. *Information cube* (Rekimoto & Green 1993) uses semi-transparent nested boxes or nested cubes to represent the hierarchical information. In detail, it presents the parent-child relationships by recursively placing child cubes inside their parent cubes that the outermost cube is the top level of data. All the cubes in this technique are transparent so that the nested sub-tree can be viewed inside the cube (see Figure 1.14). The labels are displayed on the cube surfaces. The user can inspect the detail information by rotating or moving the cube. Although, the *information cube* provides an overall view of the information, this technique does not show explicitly the relational structures of information. This costs extra cognitive effort of viewers to understand the relational structures that are presented implicitly similar to the two-dimensional *enclosure* approach.

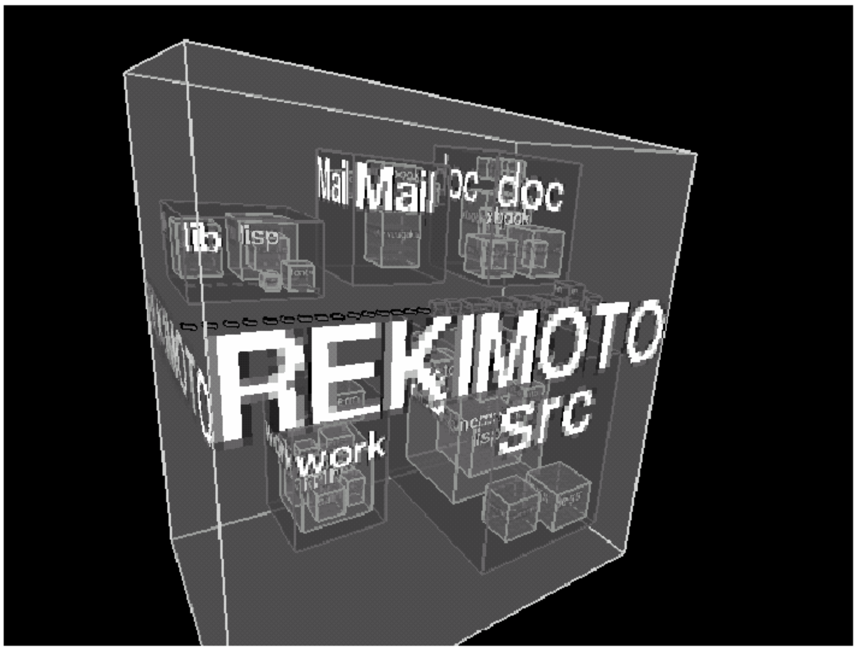


Figure 1.14. A visualisation of information cube.
Source from: (Rekimoto & Green 1993).

1.3.2 Hierarchical Navigation and Interaction Techniques

Another important step involved in the hierarchical visualisation process is navigation and interaction. Chaomei Chen (1999) stated that

> *Navigation in a hierarchical structure involves moving from one node to another, along the existing hierarchical links in the structure. When*

the size of a hierarchy becomes large, it is desirable to enable users to have easy access to contextual information, as well as local details.

One of the most important issues involved in navigation is that users can always see (or has easy access to) the contextual information. This allows the users to always perceive where they are and where they move from in the large information space during the navigation. This also assists the users decide where they should go next, and how they are interactively navigating through the information space.

To achieve easy access to both the contextual and local information in large hierarchies, there are several navigation solutions which have been developed in accommodating geometrical layout techniques. Most of the current navigation techniques can be roughly classified into three approaches: *focus+context*, *zooming+filtering*, and *incremental exploration*. *Focus+context* techniques normally provide a better solution for accessing the contextual information during navigations, and it receives a large researching effort from information visualisation researchers. We now review and discuss these approaches, especially the *focus+context* approach.

1.3.2.1 Focus+Context Techniques

Focus+context approach includes all navigation techniques that provide users with simultaneous dual views of a small focused area as well as the overall context. In other words, it can be defined as detail views of particular parts of the information which is blended in some way with a view of the overall structure of the set. This approach improves the cost structure of the information space since the overview context is retained when enlarging the focus area. Thus, this idea amplifies the users' cognition (Card, Mackinlay & Shneiderman 1999).

Since the focus area is enlarged while the context is still maintained, the presentations of *focus+context* techniques are mainly distortion-oriented. This distorted transformation can be independent or dependent of the layout algorithms. Typical distortion-oriented *focus+context* techniques in 2D are *fisheye views* (Furnas 1986; Sarkar & Brown 1994), *polyfocal display* (Kadmon & Shlomi 1978), *bifocal lens* (Spence & Apperley 1982), *perspective wall* (Mackinlay, Robertson & Card 1991), and others. *Foucs+context* visualisation might also extend the distortion to three-dimensional space. These techniques generally provide the entire context within a cube (Carpendale, Cowperthwaite & Fracchia 1997) or a sphere (Munzner 1997) where the visualisation is distorted in some way to provide both enlarged focus information as well as the reduced context information. A detail description and taxonomy of these distortion techniques are also found in (Leung & Apperley 1994).

Other approaches achieve the *focus+context* viewing by using different type of geometry or combining multiple viewing techniques. One of the popular techniques is *hyperbolic browser* (Lamping & Rao 1996) which provide the visualisation of the information using on-Euclidian geometry. (Kreuseler & Schumann 1999) also proposed a similar approach to achieve the *focus+context* visualisation, called *magic eye view*. However, this technique does not construct the layout using hyperbolic geometry, but it maps the graph layout onto a surface of a hemisphere and then it projects this image onto a plane. This technique produces an interesting result where nodes lie on a concentric circle of different radii. The focus sub-graph is enlarged by changing the centre of the projection.

The *focus+context* can also be achieved using visual cues to highlight and enhance the clarity of the focus sub-hierarchy (Herman et al. 1999). The visual cues are also used to emphasize the schematic view or the skeleton of a tree or graph. Finally, the visualisation might also combine multiple techniques, such as *focus+context*, zooming, filtering and animations to achieve their *focus+context* views, such as *cone tree* (Robertson, Mackinlay & Card 1991), *space-tree* (Plaisant, Grosjean & Bederson 2002), etc. This approach often provides the users with a better browsing mechanism allowing them to interactively move the focus area into the centre of the screen.

A few typical distortion-oriented *focus+context* techniques will now be reviewed and discussed.

Bifocal display – this technique was firstly introduced in 1982 by (Spence & Apperley) in a one-dimensional form. *Bifocal display* involves a combination of a *detailed view* and two distorted *side-views* where the *detailed view* does not have demagnification and the *side-views* is compressed uniformly in the horizontal direction (see Figure 1.15). The *bifocal display* was also extended to 2D by Leung in (1989) where the visual area is subdivided into nine sub-regions. This technique is relatively simple to implement and provides spatial continuity between regions. However, the discontinuity of magnification at the boundary between the detailed view and the distorted view is a disadvantage of this technique.

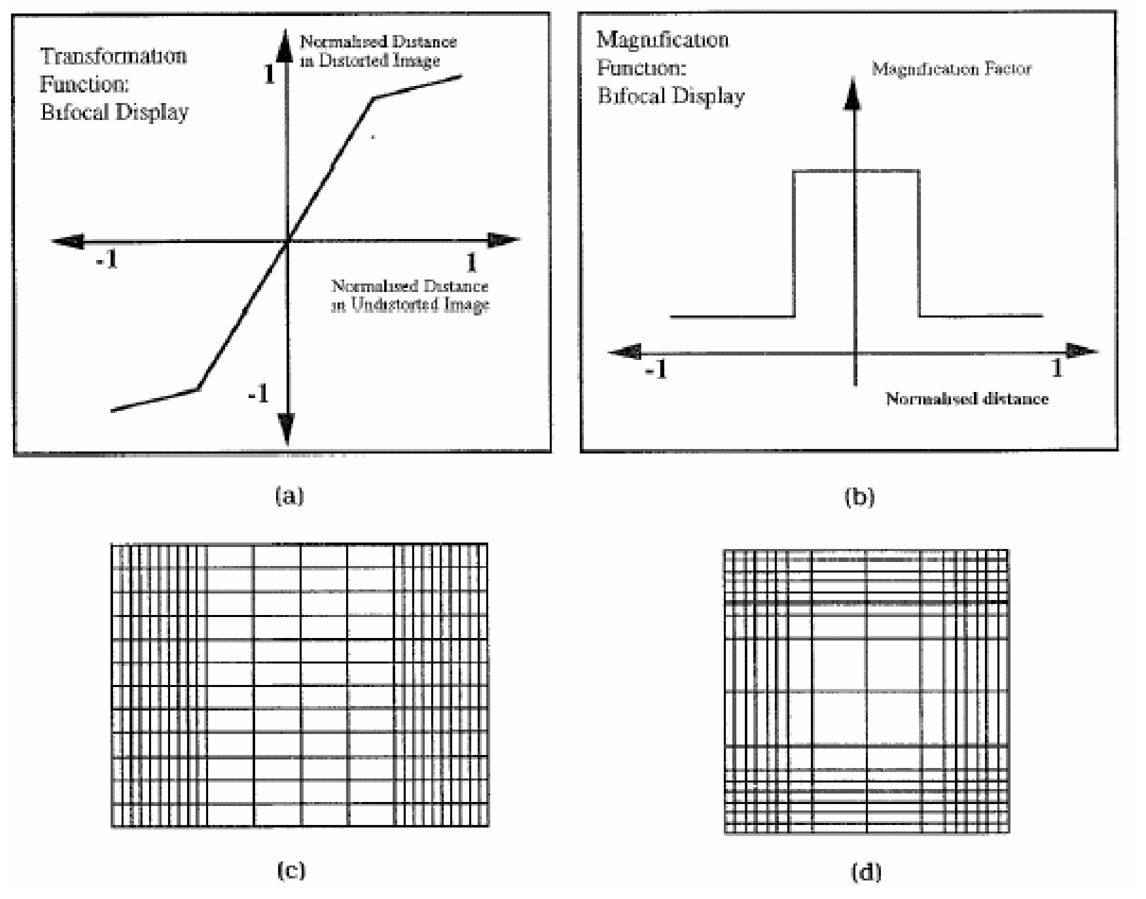

Transformation Function: Bifocal Display
Normalised Distance in Distorted Image
1
-1
1
-1
Normalised Distance in Undistorted Image
Magnification Function: Bifocal Display
Magnification Factor
-1
1
Normalised distance
(a)
(b)
(c)
(d)

Figure 1.15. The bifocal display.
(a) a typical transformation function; (b) the corresponding magnification function; (c) the application of the display in one dimension; (d) the application of the display in two dimensions. Source from: (Leung & Apperley 1994).

Perspective wall - (Mackinlay, Robertson & Card 1991) described a similar concept to *bifocal display* (Spence & Apperley 1982) which was based on the notion of smoothly integrating detailed and contextual views to assist the visualisation of linear information. The *perspective wall* is two dimensional and it includes a focus view called *middle panel* and two *side panels*. The two demagnified sides show the distorted view of the out-of-focus regions and the demagnification is proportional to their distance from the viewer. Similar to *bifocal display*, the magnification function is also discontinuous and the discontinuity is proportionally dependent on the magnitude of the angle θ between the middle panel and the side panels (see Figure 1.16).

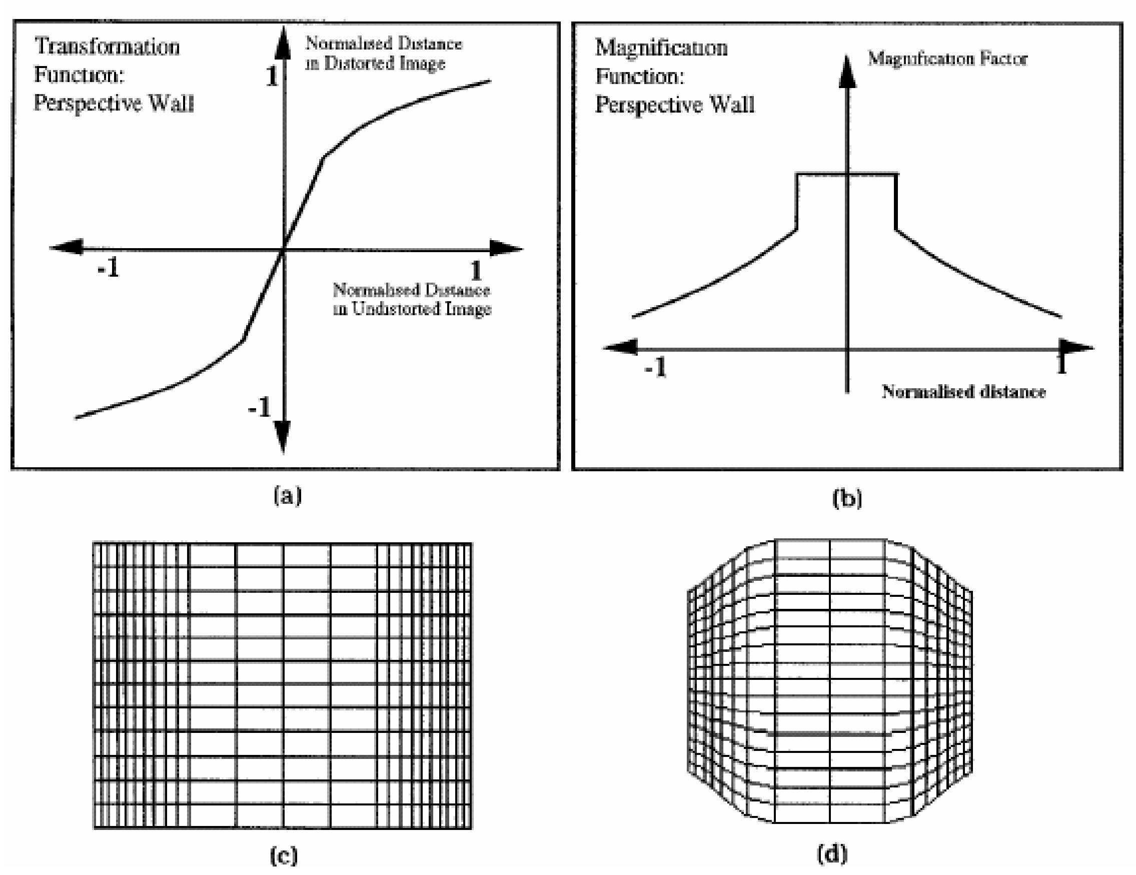

Transformation Function: Perspective Wall
Normalised Distance in Distorted Image
1
-1
1
-1
Normalised Distance in Undistorted Image
Magnification Function: Perspective Wall
Magnification Factor
-1
1
Normalised distance
(a)
(b)
(c)
(d)

Figure 1.16. The perpective wall.

(a) a typical transformation function; (b) the corresponding magnification function; (c) the application of the wall in one dimension; (d) the application of the wall in two dimensions. Source from: (Leung & Apperley 1994).

Fisheye distortion (or fisheye views) – *fisheye distortion* navigation technique was firstly proposed by Furnas in (1986) as a presentation strategy for hierarchical data structure. The concept of the *fisheye views* is to enlarge an area of interest, while other portions of the image are shown with successively less detail. There are several variations of fisheye navigation techniques including the *fisheye views* technique from (Hollands et al. 1989) and (Mitta 1990), and *graphical fisheye views* (Sarkar & Brown 1994). The *graphical fisheye views* (Sarkar & Brown 1994) includes two implementations based on a *Cartesian coordinates* transformation system and a *polar system* (see Figure 1.17). Because of the nature of these transformations, a straight line will be transformed into a curved line. This increases the complexity and implementation time of the algorithm. This problem can be solved by drawing straight lines for the connections among nodes. However, this creates unexpected edges crossing in the visualisation that decrease the quality of the display, in terms of human understanding and comprehension.

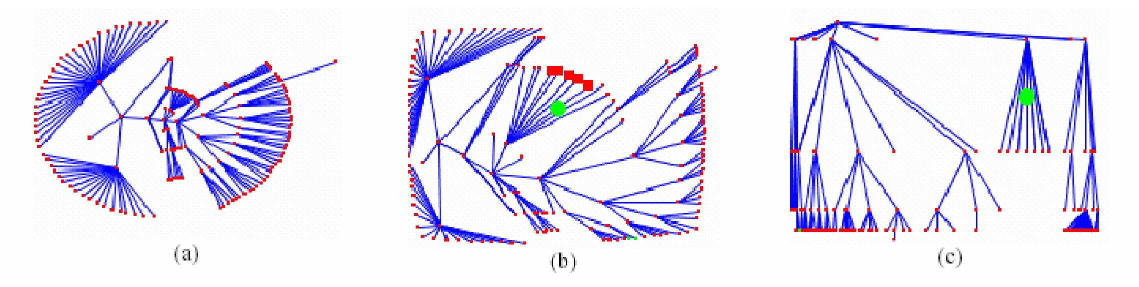


Figure 1.17. Fisheye views.

(a) the graph without fisheyes; (b) the graph with a polar fisheye; (c) the graph with Cartesian fisheye. Source from: (Herman, Melancon & Marshall 2000).

1.3.2.2 Zooming+Filtering Techniques

Zooming+filtering is defined as a viewing approach that works by reducing the amount of context in the display. The reduction is done by filtering the information in the form of selecting a subset of the data along a range of numerical values of one or more dimensions. There are two approaches for zooming including *geometric zooming* and *semantic zooming*. *Geometric zooming* is defined as the enlargement of the particular geometry area or graph or tree content without modifying the layout manner of information. *Semantic zooming* also enlarges the focus section, but the information content also changes so that more detail is shown at a particular sub-graph or sub-hierarchy. Although *zooming+filtering* is a natural way to navigate through a graph or tree hierarchy, this technique also suffers from the limitation of loss of context information during the navigation. The lost of context content makes it much harder for a user to navigate through a large information space.

There are several *zooming+filtering* techniques available, and they are generally dependant on the layout algorithms. Typical *zooming+filtering* techniques are *starfield display* (Jog & Shneiderman 1995), *tree navigation system* (Herman, Delest & Melancon 1998), *tree-maps* (Johnson & Shneiderman 1991), *Pad* (Perlin & Fox 1993), *Pad++* (Bederson et al. 1996), etc. (van Wijk & Nuij 2003) also presented a novel smooth and efficient zooming and panning technique for transferring one view to another on two-dimensional visualisation models.

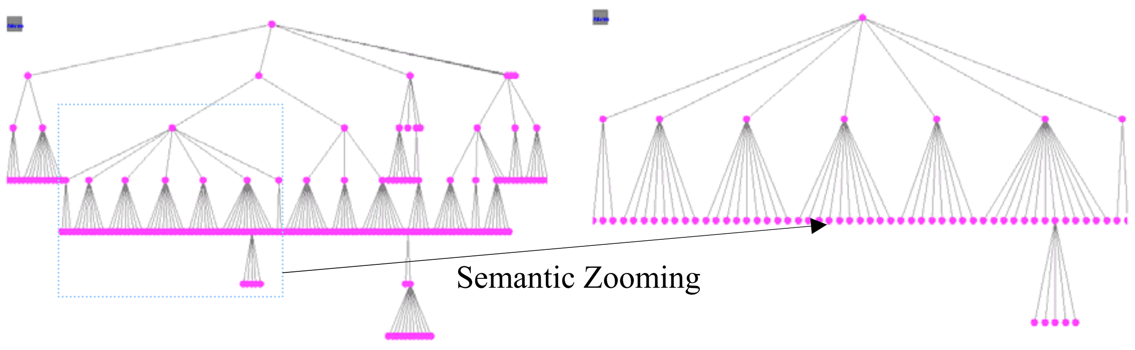


Figure 1.18. An example of semantic zooming of classical hierarchical layout.

1.3.2.3 Incremental Exploration

Incremental exploration is defined as a viewing approach that displays only a small portion of the full graph/tree incrementally following the user's exploration of information space (see Figure 1.19). Thus, these techniques are able to handle huge datasets where it is impossible to display the entire hierarchy on the screen at a time. Although this approach is relatively new, it already produces some interesting results from experiments. Typical incremental exploration techniques are described at (Huang & Eades 1998), (SC North 1995), (Brandes & Wagner 1997), etc.

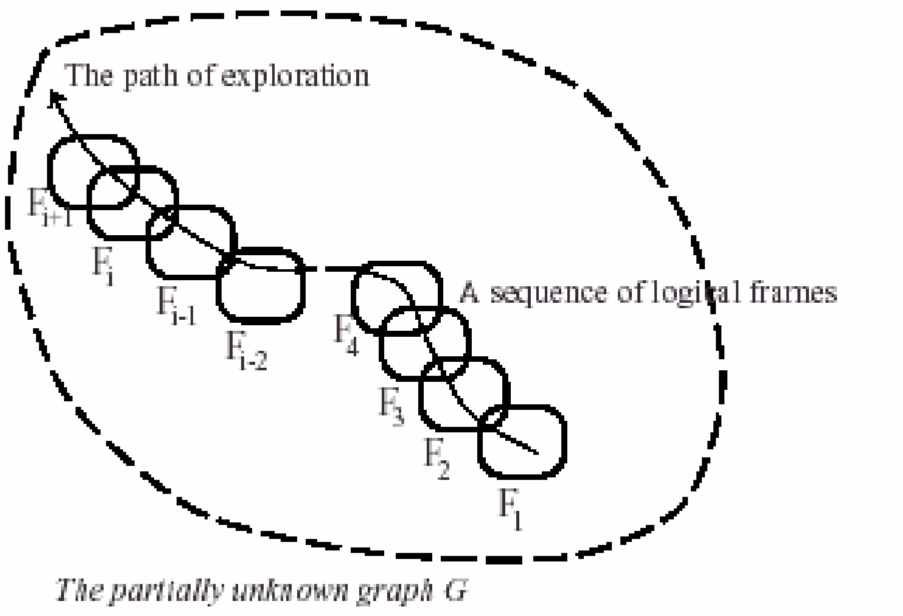


Figure 1.19. An example of incremental exploration of huge graph.
Source from: (Huang, Eades & Cohen 1998).

1.3.3 Other Information Visualisation Approaches

There are several layout and navigation techniques for visualising hierarchies or graphs in both two-dimensional and three-dimensional space which do not belong to any of the above classifications. Each technique is normally suitable or best performed on a particular application. More detail about layout and navigation algorithms can also be found at (Card, Mackinlay & Shneiderman 1999), (Spence 2001), (Herman, Melancon & Marshall 2000), (Chi 2002), and (Chen 1999).

Layout techniques can also use force, or called *spring-mass* model, to define the positions of nodes and edges. In these techniques, each node may be given individual weight and the edge is given strength. Several works in this direction have been proposed and implemented in both two-dimensional geometry and three-dimensional space. Popular 2D graph layout algorithms in this approach were presented by (Eades 1984), (Eades 1984; Eades & Huang 2000), (Gansner & North 1998), (Cheswick, Burch & Branigan 2000), and others. Examples of 3D force-directed graph layout algorithms are from (Osawa 2001), *3D narcissus* (Hendley et al. 1995), and others. The *spring-mass model* is even extended to a non-Euclidean geometry (Kobourov & Wampler 2004). *Force-directed* algorithms are simple and easy to implement, modify, and the layouts are reasonably nice overall (see Figure 1.20). However, these techniques have very high computational times which might make them unsuitable for visualising the entire large hierarchies.

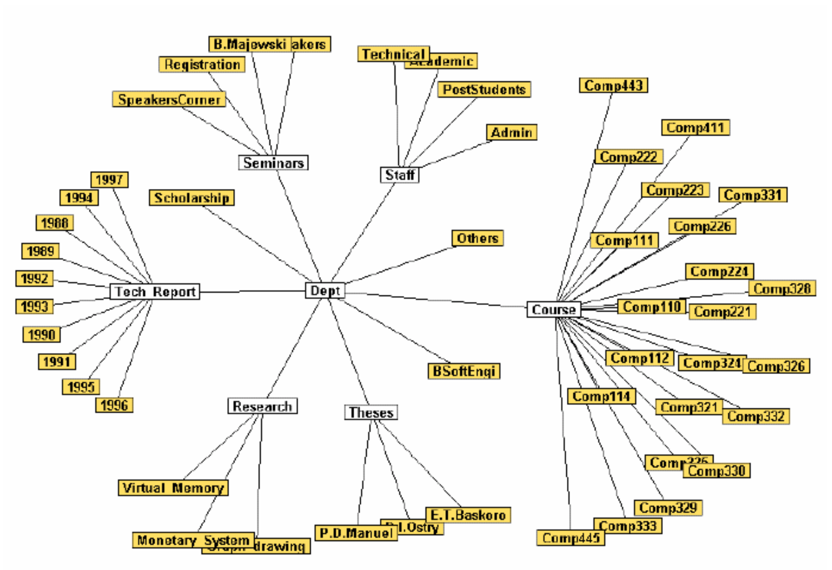


Figure 1.20. A visualisation of spring-mass model.
Source from: (Huang, Eades & Cohen 1998).

In order to avoid the visual complexity and density of the layout, some layouts use a *clustering* metaphor for their layout and navigation (Feng 1997; Kreuseler & Schumann 1999). These methods attempt to discover clusters within the data and reduce the number of elements to display by restricting the view to the clusters themselves. This provides an overview of the structure and allows viewer to retain a context while reducing visual complexity. A common approach is to present the clusters with glyphs and treat them as super-nodes in a higher-level or compound graph which the user can navigate instead of the original graph. Huang and Eades (1998) defined an alternative approach, called *abridgement*, which allows edges between super-nodes to be induced. The detail of clustering algorithms can be found in Feng's Ph.D. thesis (1997).

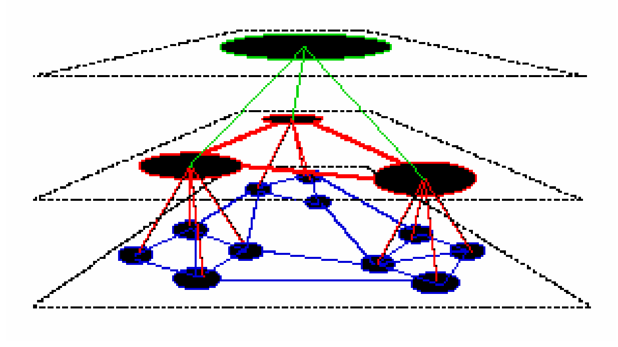


Figure 1.21. A structure induced by hierarchical clustering.
Source from: (Eades & Feng 1996).

1.3.4 Limitations of Existing Hierarchical Visualisation Techniques

With the rapid growth of information, the size of datasets increases significantly each year. The traditional user interfaces of the information system, with the typical working mode of textual display plus scrolling bar, have no longer meet the satisfaction of users in terms of human cognitive process. It is very time consuming for these users to read through the text line by line through the whole page or several pages in order to find information they need. Furthermore, it is even hard for them to extract some inter-relationships among the texts of one page or across several pages. Therefore, many designers of the information systems have realised that the development of new visual interface that can reduce the human cognitive cost for their next generation products will be the key to the success.

Since the size of datasets is often very large with thousands or even millions of items, the traditional visual user interfaces are unable to cope well with the display of large visualisation on a standard monitor with the size around seventeen to nineteen inches. This is because that many of the existing visualisation methods use the *virtual page* solution to this problem. That is, the graph is laid out on a virtual (very large) page, and then a small window and scroll bars are provided to allow users to navigate the visualisation. The main limitation of this solution is that the overall visualisation or the global view of the information space is broken into several windows. This makes it difficult for users to understand the scope of information spaces or to find where they are and where they should go to access information they need. Furthermore, this approach needs a huge memory to store and display the large virtual screen.

In addition to above, the use of an expensive large and high resolution display does not guarantee the significant improvement of visualisation compared to a standard display (Ware 2004). This is because of the constraint limits of our brain pixels.

This section discusses specifically the limitations of existing hierarchical visualisation techniques which include both layout techniques as well as navigation and interaction techniques.

1.3.4.1 Limitations of the Layout Techniques

Although those proposed information visualisation techniques, as described at section 1.3.1, can be used to deal with hierarchical visualisation, only a few are good candidates for visualising large hierarchies on a normal personal computer. The limitations of those techniques can be summarised:

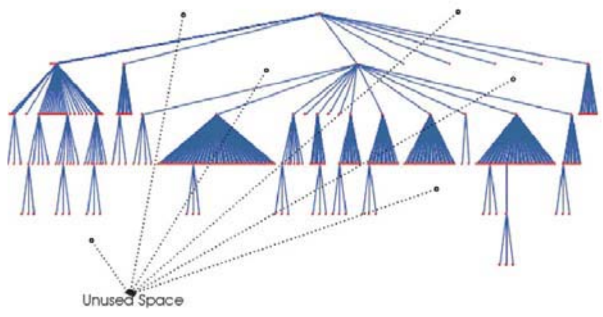
- The *connection* approach, such as *hyperbolic tree*, *radial view*, *balloon view*, *classical hierarchical tree*, etc, uses node-link diagrams to present relationships among data. These techniques allow the viewer to directly see the relationships among data items as a set of graphical edges in the diagram. However, most layouts produced by the *connection* approach inefficiently use of display spaces and they usually contain many unused spaces that waste the display space. Thus this reduces significantly the number of nodes and edges to be displayed on the screen (see Figure 1.22). There are some typical connection based techniques for visualising large hierarchies, such as *hyperbolic browser*, *NicheWorks*, and *Internet mapping*. The limitations of these techniques are further described below:

  - *Hyperbolic browser* (Lamping & Rao 1996) is an excellent focus+context technique for browsing large hierarchies. This technique, however, does not use space efficiently so that it only displays information inside a circular disk or a sphere and it has a large portion of unused space. In addition, the algorithm is complex and hard to understand due to its different geometrical background.
  - *NicheWorks* (Wills 1999) overall gains better space efficiency compared to the *hyperbolic browser*, but this technique only supports zooming so that the context might be lost during navigation.
  - *Internet mapping* (Cheswick & Burch 2002) is highly capable for very large tree-like structure (100,000 nodes or more). However, this technique is very slow so that a typical run requires 20 CPU hours on a 400 MHz Pentium.

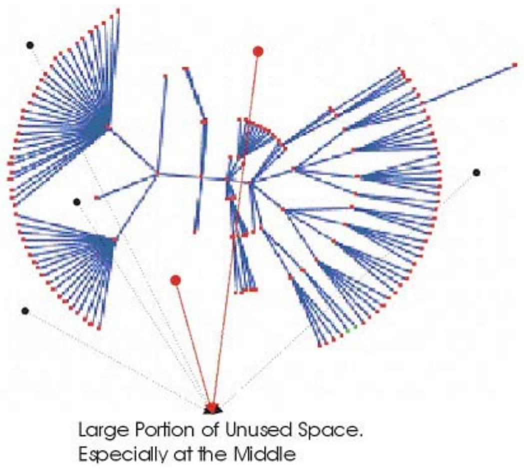
- Enclosure techniques, typically *tree-maps* (Johnson & Shneiderman 1991) and its variations (Bruls, Huizing & van Wijk 2000; van Wijk & van de Wetering 1999) claim 100% efficiency of space utilisation. However, they do not show directly the relational structures of information by providing edge-links to connect nodes. This costs extra cognitive effort for viewers in understanding the relational structures that are presented in an enclosure manner (see Figure 1.10).

Thus, investigating a new approach to overcome the above limitations of both approaches is essential for efficient visualisation of large hierarchies. The expected layout algorithms need to provide not only the solutions for optimising display space allowing more information to be displayed in the limited display space, but it also retains reasonable clarity or quality of the visualisation.

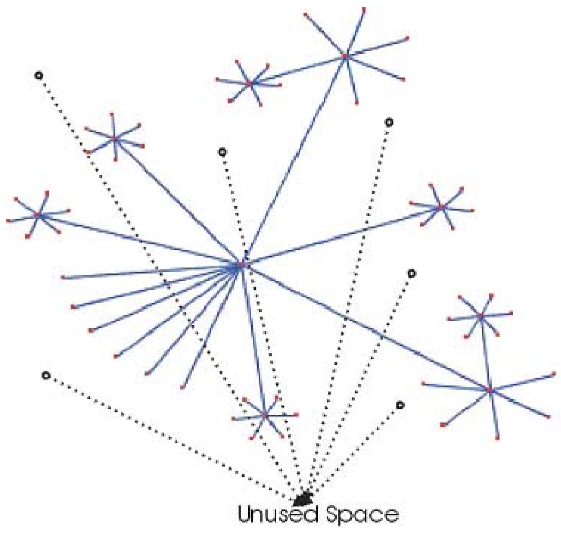
Three-dimensional visualisation methods such as *cone trees* (Robertson, Mackinlay & Card 1991), *botanical tree* (Kleiberg, van de Wetering & van Wijk 2001) and *3D hyperbolic browser* (Munzner 1997) increase the density of information of the screen by extending an extra dimensional of visual display space. However, this approach requires extra computational costs and possible special hardware and software supports, such as high speed graphic workstations with high display resolution, which are difficult to be applied in most ordinary personal computers. Furthermore, three-dimensional techniques might suffer from the occlusion problem if a three-dimensional metaphor is presented and navigated in a two-dimensional display medium.



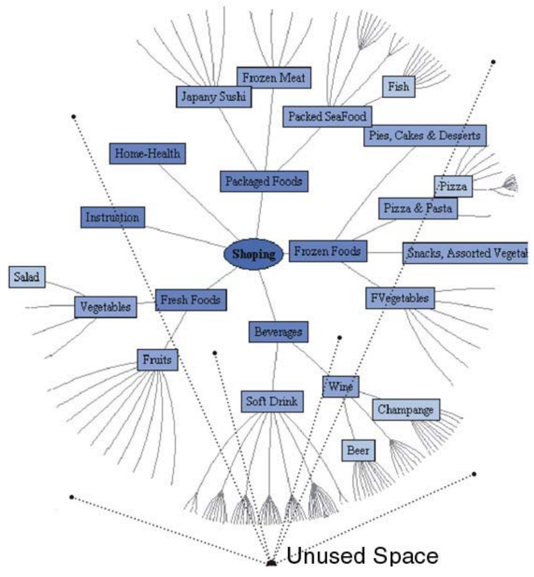
(a) Classical Hierarchical View



(b) Radial View



(c) Balloon View



(d) Hyperbolic Browser

Figure 1.22. The inefficiency of space utilisation techniques in *connection* approach.

1.3.4.2 Limitations of the Navigation and Interaction Techniques

As mentioned in section 1.3.2, there are three approaches for the navigation and interaction of hierarchies. In comparison to the above approaches, the *focus+context* approach normally provides a better solution for accessing the contextual information during navigation. To achieve this, it uses the *enlarge+embedded* or *enlarge+blended* concept (see Figure 1.23), that usually uses a particular graphic technique such as *fisheye distortion* or *semantic zooming* to enlarge a portion of information structure to form a *detail view*. Therefore, we sometimes call this *detail view* as *focus view*. It then embeds this *focus view* into the *global view* of the overall contextual information structure for visualisation. This approach is one of the most efficient and effective navigation strategies in the current interactive visualisation design, and is widely used and commercialised in many visualisation and Graphic User Interface (GUI) tools. However, one obvious limitation of this approach is the area division, that is the whole geometrical area has to be divided into two parts; one for the drawing of local structure and another for the drawing of the rest of the global structure (see Figure 1.23). This, therefore, limits the display of details in the focus view.

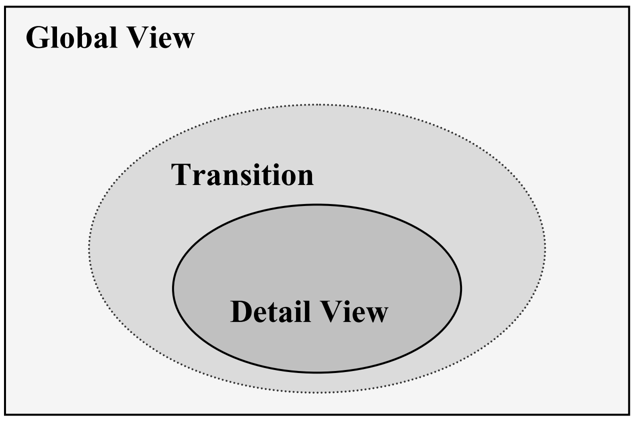


Figure 1.23. The traditional *focus+context* concept.

Although many *focus+context* viewing techniques that have been developed and have worked well with medium size information spaces, the exploration of large information spaces becomes increasingly difficult as the volume of data grows. Most of the problems are related to the limited screen space of the display. A number of the existing *focus+context* techniques require the division of display space for displaying both global and detail views, such as (Andrews & Heidegger 1998), (Brandes & Wagner 1997), (Cheswick, Burch & Branigan 2000), and others. Similarly, some other techniques also attempt to provide both global and detail views by splitting the display area or providing extra windows, typically called *multiple-views* techniques (Baldonado, Woodruff & Kuchinsky 2000; Convertino et al. 2003; C North 2001; Robert 1998). Thus, the available space left for displaying the large and complex global structure is getting even smaller.

Despite the potential of current *focus+context* navigation techniques, this approach is still an open problem for further improvement. Thus investigating an alternative *focus+context* technique that can take advantage of the current focus+context viewing scheme, but avoid the limitation of the "area division" concept is desirable. The expected new *focus+context* navigation technique should increase the area of *focus view* allowing more information to be displayed in this view while retains the quality of the *context view*.

1.4 Challenges

The major problems with the current systems for visualising and navigating large hierarchies may be outlined as below:

- Node-link diagrams use the display space inefficiently: a large portion of screen pixels are used as background (see Figure 1.22). Therefore, existing systems are seldom able to display a *global view* of medium or large size information spaces on one screen for navigation. For example, they are unable to display a complete on-line file structure of Java JDK v.1.4.1 documentation with approx. 9,500 directories and files on one screen. This blocks users from gaining an overall view of the entire file system, and navigating the file system in a single screen without scrolling. A split view can show only part of the information space which may not contain what is needed. Furthermore, the currently visible part in most cases does not indicate where the user should go to locate particular data items. Scrolling among split views will break a user's mental map of the global view and create extra cognition overheads to identify connections between views. Thus, the first challenge of this research is: *how to visualise large relational hierarchies efficiently on a limited display space without losing its visualisation quality?*

- The computation of layouts of large node-link diagrams with thousands of nodes is very expensive. Most existing graph layout algorithms have super-linear time complexity, and in practice are too slow for the calculation of geometry if the number of nodes is larger than a few hundred. Thus, the second challenge of this research is: *how to find out an optimised layout algorithm that can complete the computation of geometrical layout of large hierarchies in minimal time, i.e. in linear time?*
- Traditional *focus+context* techniques use the *area division* concept (see Figure 1.23) to calculate the focus view. They then embed the focus view into the global view for displaying information spaces. This limits the size of screen space allocated for displaying the focus view. It means that the amount of information that can be displayed with more details in the focus view is limited. Thus, the third challenge of this research is: *how to navigate through large relational hierarchies effectively to access information they need?*

This thesis proposes new layout and navigation techniques for the visualisation of large hierarchies due to these challenges. Section 1.6 describes the author's contributions in the area of computer science.

1.5 Research Objectives

The overall objective of this research is to investigate visualisation techniques which address all of the problems mentioned in the *challenges* section. More specifically my research objectives are described below:

- To define a new visualisation approach that supports the space-efficient visualisation of large relational hierarchies, which can be displayed on ordinary PCs with limited screen size and computational power.
- To investigate efficient layout algorithms which enable the display of many items on a screen by maximising the utilisation of geometrical spaces, while retaining good readability of the visualisation.
- To investigate a navigation method that can produce a series of navigational focus views that can display more detailed information.

- To investigate a navigation method that can simplify fisheye distortional views so that reduce the computational cost during the navigation.
- To conduct an experimental evaluation of the above layout algorithms for aesthetics and efficient space utilisation.

1.6 Author's Contributions in the Thesis

The overall contribution of this thesis is the introduction of a new hierarchical visualisation approach, *space-efficient visualisation*, which is the first attempt to use a space-efficient approach to address the problem of visualising large hierarchies. This approach differs from other approaches, such as the *virtual page* and the *three-dimensional* approaches. It is one of the most cost-effective ways to solve the problem, by which the extra hardware and software requirements for visualising large hierarchies are minimised.

Specific contributions of this thesis are:

- A new type of hierarchical visualisation techniques, called *enclosure+connection*, that inherits the advantages of both *enclosure* and *connection* approaches.
- Two space-efficient layout algorithms, *space-optimised tree and EncCon tree* that use the area partitioning approach to lay out the node-link diagrams. Both algorithms can generate hierarchical layouts with not only the efficient utilisation of display space, but also a direct perception of the hierarchical relationships.
- A new *focus+context* navigation method, called *hybrid view* that simplifies the fisheye distortion to speed up the navigation.
- A new *focus+context* navigation method, called *layering view* that uses transparent graphics to achieve the dual-layer display of information. This approach enables not only a full display of *focus view* but also a display of *context view*.

- Two practical systems have been developed as case studies, including a *visual interface in shared collaborative workspaces* and a *visual browser for large-scale online auctions*.
- The author has also carried out an experimental evaluation to justify the effectiveness of the *EncCon tree* layout algorithm based on the aesthetic rules of graph drawing.
- Finally, this thesis proposes a possible extension of our algorithms to three-dimensional space, and an effective technique for the navigation of classical hierarchical layout.

This thesis covers the research results obtained by the author when studying for his Ph.D. Most of the results have been published as research papers in journals and refereed conference proceedings (see section *Author's Publications for the Ph.D*). The details of author's contributions will be presented in the following chapters.

1.7 Thesis Organisation

This thesis is organised by chapters as follows:

Chapter 2 describes the technical detail of a new space-efficient hierarchical visualisation including an *enclosure+connection* visualisation model and two geometrical layout algorithms, called *space-optimised tree* and *EncCon tree*. This chapter discusses the computational complexity of both algorithms as well as provides examples of the visualisations resulting from the layout algorithms. An experimental evaluation is also presented in this chapter to justify the effectiveness of the *EncCon tree* layout algorithm against the popular *tree-maps* algorithms, especially *squarified tree-maps*.

Chapter 3 describes the technical detail of two new interactive navigation methods, namely *hybrid view,* and *layering view*. This chapter first describes the *hybrid view* technique which includes semantic zooming, browsing and distortion algorithms. It next presents the technical detail of the *layering view* technique which includes layering display, interactive navigation and animation.

Chapter 4 shows the detail of a practical application which has been developed as a case study. This application is a visual interface in shared collaborative workspaces and a visual browser for large-scale online auctions.

Chapter 5 also describes another practical application in e-commerce. The system applies layout and navigation algorithms for browsing and analysing product catalogues of large-scale online auctions.

Chapter 6 describes a preliminary work of the extension of the layout algorithms to three-dimensional space. This chapter also shows an effective technique for the navigation of classical hierarchical layout.

Chapter 7 concludes this thesis by giving a summary of the points presented, the contributions made, and the possibility of future work.

Chapter 2 Space-Efficient Visualisation

This chapter describes the technical detail of a new space-efficient hierarchical visualisation including an *enclosure+connection* visualisation model and two geometrical layout algorithms, the *space-optimised tree* and the *EncCon tree,* that are the core of this new visualisation. Essentially, the *enclosure+connection* visualisation uses the area partitioning to recursively lay out the entire hierarchy while a node-link diagram is also used to present the hierarchical relationships explicitly. The layout algorithms used in this visualisation can maximise the utilisation of display space. These layout algorithms calculate the position of a node-link diagram based on its weight property.

This chapter also discusses the computational complexity of both layout algorithms and some example layouts generated by these algorithms. A comparison of area partitioning algorithms between *EncCon tree* and *tree-maps* (including *squarified, strip* and *slice and dice tree-maps*) is then discussed. Finally, we present an experimental evaluation of two 'square-like' partitioning algorithms: *EncCon tree* and *squarified tree-maps*. This evaluation is conducted based on four typical aesthetic rules of graph drawing including *edge crossings*, *angular resolution*, *total edge length*, and *uniform edge length*.

2.1 The Enclosure+Connection Visualisation Model

The *enclosure+connection* visualisation takes the advantages of both the ordinary *enclosure* and *connection* visualisation approaches. Particularly, it inherits the *area partitioning* technique from the *enclosure* approach to ensure the maximum utilisation of geometrical space for displaying trees, while still uses a node-link diagram to represent the relational structures explicitly to reduce the cognition overhead. The steps of constructing such visualisation can be described as below:

- **Step1** – calculate an associated weight for every vertex v in the tree T. A weight is a value associated with the property of a vertex. The weight of a vertex might be dependent to the domain-specific attributes of that vertex such as the size of a file or an object, the role of a person in an organisation, etc. In this thesis, the weight of a vertex is simply defined based on the numbers of its descendants. The system calculates the weights of all vertices before starting the geometrical partitioning of the hierarchy.

- **Step 2** - recursively partition the entire display area into a set of sub-display areas, called *local regions*, and then assign these *local regions* to every vertex in the tree *T*. Each vertex v_i is bounded by a local region $R(v_i)$ and the drawing of the sub-tree $T(v_i)$ is restricted to inside the geometrical area of $R(v_i)$. Therefore, the local region $R(v_i)$ of vertex v_i is the sum of the areas assigned to its children. The size of a local region $R(v_i)$ is depends on the value of weight associated with the vertex v_i.
- **Step 3** - position all vertices and their sub-trees inside their display local regions. In most cases, the position of a vertex is simply located at the centre of its local region. The layout algorithm, which is responsible for positioning of all the vertices $\{v_1, v_2, ..., v_n\}$ of the given tree *T* in a two-dimensional geometrical space, is governed by a particular area partitioning algorithm described in step 2.
- **Step 4** - assign the graphical attributes, including the *size*, *label*, *shape*, and *colour* to all vertices and edges based on their levelling property in the hierarchy. Under the *enclosure+connection* visualisation scheme, the local region assigned to a child vertex is always smaller than the one assigned to its parent. Therefore, in order to display the complete hierarchical structure with thousands of vertices and edges in a limited display space, we assign different graphic properties to vertices and edges at different levels of the hierarchy. For example, the closer to the root, the larger size of graphical nodes and the wider of links are. This rule improves the clarity of the presentation of hierarchical structure by avoiding overcrowded and overlaps among graphical nodes and links. We then display the visualisation on the screen.

2.2 Weight Calculation

We assign a weight *w(v)* to each vertex *v* for the calculation of the local region *R(v)* that relates to the regions of its father and siblings. There are many ways to define the weight of a vertex, and one way to do it is based on the vertex property. This thesis, however, defines the weight of a node simply based on its descendant. The calculation of nodes' weights is independent to the layout algorithms and this process starts before the geometrical partition. Weights of vertices are calculated before starting the partitioning of geometrical layout. The layout algorithms of both *space-optimised tree* and *EncCon tree* will then partition the geometry based on the weights of nodes.

Each vertex v_i in tree T is associated with a weight $w(v_i)$ and thus we have a set of weights $\{w(v_1), w(v_2), ..., w(v_n)\}$ associated with the vertex set $\{v_1, v_2, ...,v_n\}$ in T. The set of vertex's weights can be calculated recursively from leaves in the following rules:

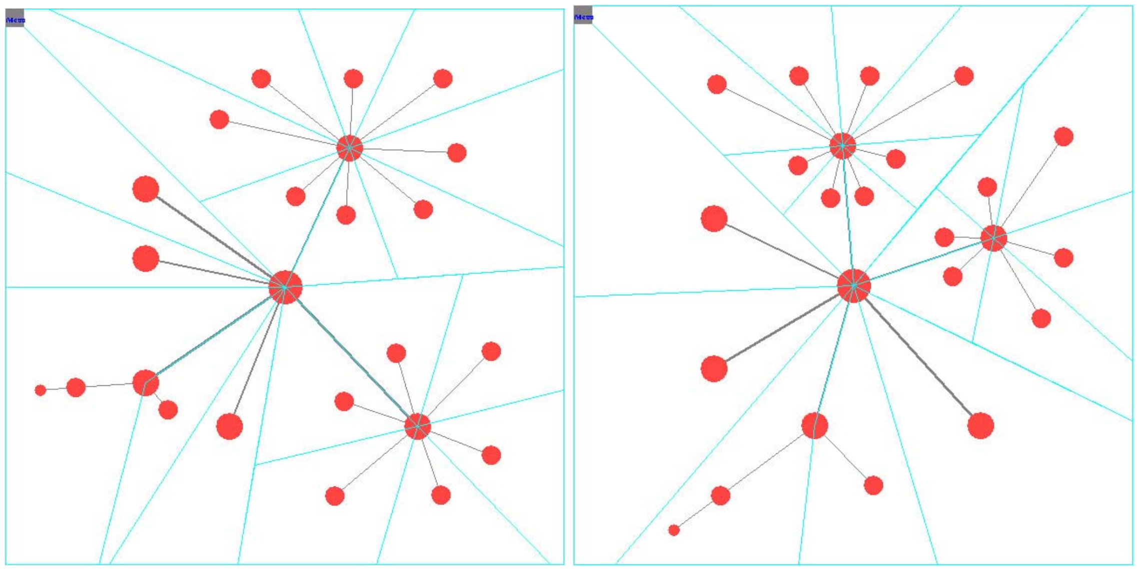
- If a vertex v is a leaf, its weight is defaulted as $w(v) = 1$.
- If vertex v has k children $\{v_{l+1}, v_{l+2}, ..., v_{l+k}\}$, its weight is calculated by using formula:

$$w(v) = 1 + C\sum_{i=1}^{k} w(v_{l+i})$$

Equation 2.1

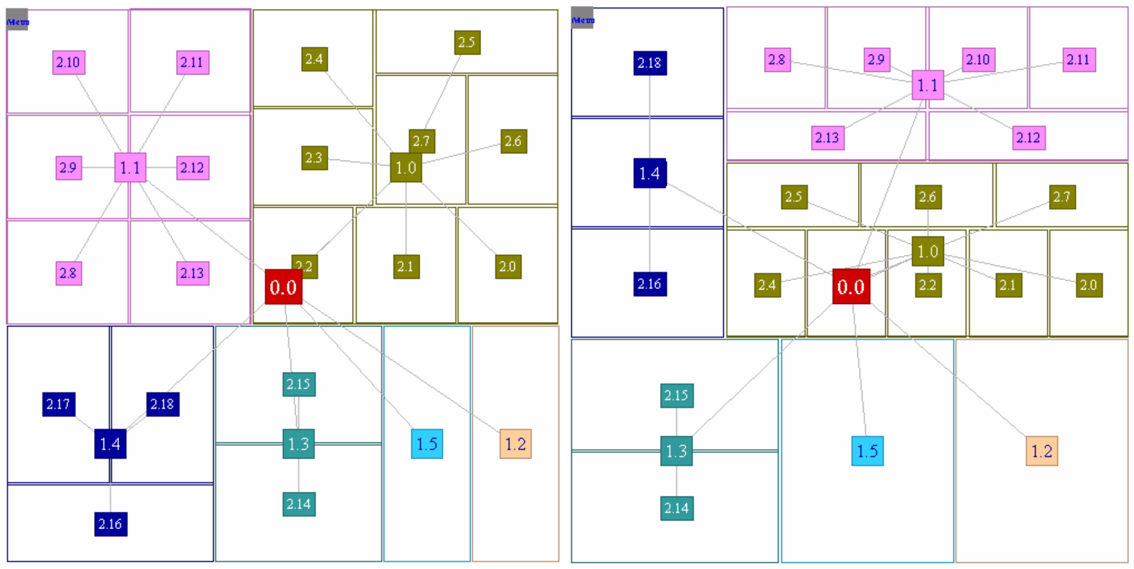
where C is a constant ($0 < C < 1$), and $w(v_{l+i})$ is the weight assigned to the ith child of vertex v. Constant C is a scalar that determines the difference of local regions' sizes based on the number of descendants of these vertexes. In other words, the larger the C's value is, the bigger the difference of local regions $R(v)$ of vertices with more descendants and vertices with fewer descendants. Figure 2.1 is an example of *space-optimised tree*'s area division with a small dataset, using different C's values. Similarly, Figure 2.2 also shows an example of *EncCon tree's* area division with another small dataset, using different C's values. We can see that in Figure 2.1a and Figure 2.2a, the differences among local regions are much larger than those in Figure 2.1b and Figure 2.2b, respectively.

The value of constant C can be adjusted to ensure the best performance in each application. In general, the layout-optimised tree performs well with C = 0.6 for the first approach and C = 0.45 for the second approach. The *EncCon tree* layout algorithm produces overall good results with C = 0.45.



(a) (b)

Figure 2.1. An example of *space-optimised tree's* area partitioning. Using (a) C = 0.6 and (b) C = 0.1.



(a) (b)

Figure 2.2. An example of *EncCon tree* area partitioning. Using (a) C = 0.45 and (b) C = 0.1.

2.3 Space-Optimised Tree

The geometrical drawing $D(T)$ of a tree T is responsible for the positioning of a set of vertices $\{v_1, v_2, ..., v_n\}$ and the computation of polygonal local regions $\{R(v_1), R(v_2), ..., R(v_n)\}$ for the drawing of sub-trees $\{T(v_1), T(v_2), ..., T(v_n)\}$ in a two-dimensional plane.

In general, the *space-optimised tree* uses enclosure approach to partition the geometrical display in which each vertex v_i is bounded by a polygonal local region $R(v_i)$. The area of $R(v_i)$ is equal to the sum of the areas of its children. The drawing of a sub-tree $T(v_i)$ is restricted within the boundary of the polygon $R(v_i)$ (see an example in Figure 2.1). A geometrical layout of sub-tree $T(v_i)$ rooted at v_i is calculated based on the weight $w(v_i)$ of v_i and its local region $R(v_i)$.

We initially define the local region $R(v_r)$ for root vertex v_r as the entire rectangular display area, and then we position root v_r at the centre of $R(v_r)$. Then, all other local regions $\{R(v_1), R(v_2), ..., R(v_n)\}$ are calculated in sequence from the root outward to the leaf vertices. Suppose that we want to calculate the region $R(v_i)$ of the sub-tree $T(v_i)$ rooted at v_i which has k children $\{v_{l+1}, v_{l+2}, ...,v_{l+k}\}$. The geometrical portioning of these child vertices is generally defined by the process:

- We first calculate local regions $\{R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k})\}$ for the children $\{\{v_{l+1}, v_{l+2}, ...,v_{l+k}\}\}$.
- We then calculate positions of vertices $\{v_{l+1}, v_{l+2}, ...,v_{l+k}\}$ that are inside their local regions $\{R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k})\}$.
- We repeat the above steps to all sub-trees from the top to the bottom of the tree hierarchy and stop when all leaves of the tree are reached.
- We ignore the layout calculations for those sub-trees when local regions of these sub-trees are too small to be displayed by the current screen resolution.
- We eventually get $R(v_i) = R(v_{l+1}) \cap R(v_{l+2}) \cap ...\cap R(v_{l+k})$.

There are two approaches for the local area division in the *space-optimised tree*. The detail of both approaches is described in the following sections.

2.3.1 Local Area Partition – Approach 1

Suppose that a vertex v_i has k children $\{v_{l+1}, v_{l+2}, ..., v_{l+k}\}$ located in a local region $R(v_i)$. The polygon $R(v_i)$ is already defined and we want to divide $R(v_i)$ into sub-regions $\{R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k})\}$ for the drawings of sub-trees $\{T(v_{l+1}), T(v_{l+2}), ..., T(v_{l+k})\}$. Wedges are used to calculate the local regions of vertices as well as their geometrical positions. In this approach, a wedge is defined and calculated in the following subsections:

2.3.1.1 Wedge Calculation

Formally, the wedge of the child vertex v_{l+m} is defined as $wg(v_{l+m}) = \{v_i, l, \alpha(v_{l+m})\}$ where v_i is the father of v_{l+m} and l is a straight line going through v_i (that determines the boundary of vertex v_{l+m} and its nearest sibling) and anti-clockwise angle $\alpha(v_{l+m})$ of the wedge (see Figure 2.3).

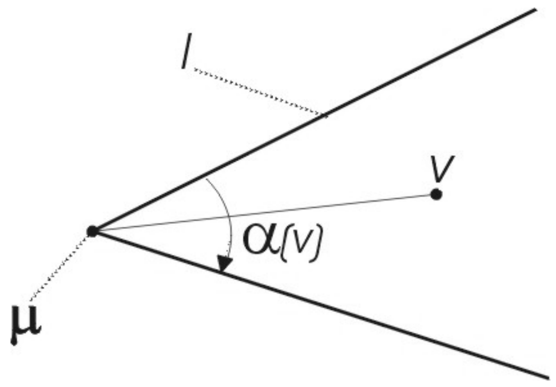


Figure 2.3. A wedge *wg(v)*.

The angle $\alpha(v_{l+m})$ of the wedge $wg(v_{l+m})$ of the mth child is calculated by the formula below:

$$\alpha(v_{l+m}) = A\frac{w(v_{l+m})}{\sum_{j=0}^{k} w(v_{l+j})}$$

Equation 2.2

where A is a constant and $w(v_{l+j})$ is the weight associated with vertex v_{l+j}. The value of A can be adjusted to produce the variations of layout. However, the prototype assigns the entire circular degree to A, i.e. A = 360, in all its demonstration.

We repeat the above calculation to get a set of wedges $\{wg(v_1), wg(v_2), ..., w(v_n)\}$ associated with every vertex in tree *T*. These wedges determine the area division of local regions which is described in the next section.

2.3.1.2 Local Region Division

A local region $R(v_{l+m})$ of the m^{th} child vertex of v_i consists of a wedge $wg(v_{l+m})$ and one (or more) cutting edge(s) intersects with the boundary of polygonal local region $T(v_i)$ (see Figure 2.4). Thus, $R(v_{l+m})$ of vertex v_{l+m} is calculated as soon as the wedge $wg(v_{l+m})$ is defined. The local region $R(v_{l+m})$ contains the drawing of a sub-tree $T(v_{l+m})$ and a directed edge (v_i, v_{l+m}) where v_i is the father of v_{l+m}.

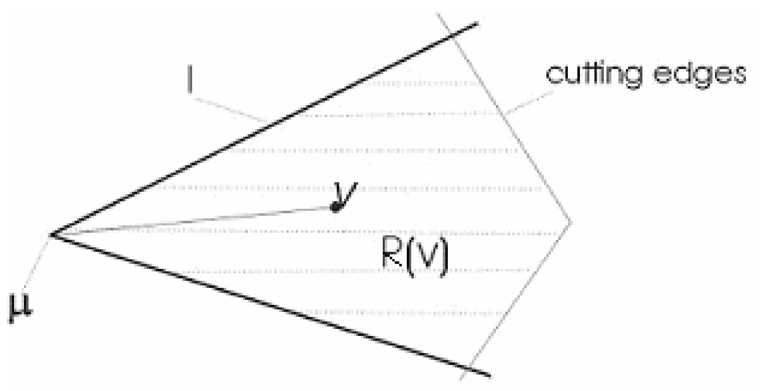


Figure 2.4. A local region *R(v)*.

Suppose that $R(v_i)$ is the local region of vertex v_i, and v_i has k children $\{v_{l+1}, v_{l+2}, ..., v_{l+k}\}$. The wedges of these child vertices are respectively $\{wg(v_{l+1}), wg(v_{l+2}), ..., wg(v_{l+k})\}$. The local regions of the child vertices $\{R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k})\}$ are respectively polygons that are formed by the intersection of $\{wg(v_{l+1}), wg(v_{l+2}), ..., wg(v_{l+k})\}$ and $R(v), R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k-1})$. Figure 2.5 shows an example of dividing v_i's local region into four sub-regions for its children $\{v_{l+1}, v_{l+2}, v_{l+3}, v_{l+4}\}$. The local region $R(v_i)$ of v_i is a pentagon and $\{\alpha(v_{l+1}), \alpha(v_{l+2}), \alpha(v_{l+3}), \alpha(v_{l+4})\}$ are respectively the angles of wedges $\{wg(v_{l+1}), wg(v_{l+2}), wg(v_{l+3}), wg(v_{l+4})\}$. This figure also indicates that the weight of vertices increases in order respectively $\{v_{l+4}, v_{l+1}, v_{l+2}, v_{l+3}\}$.

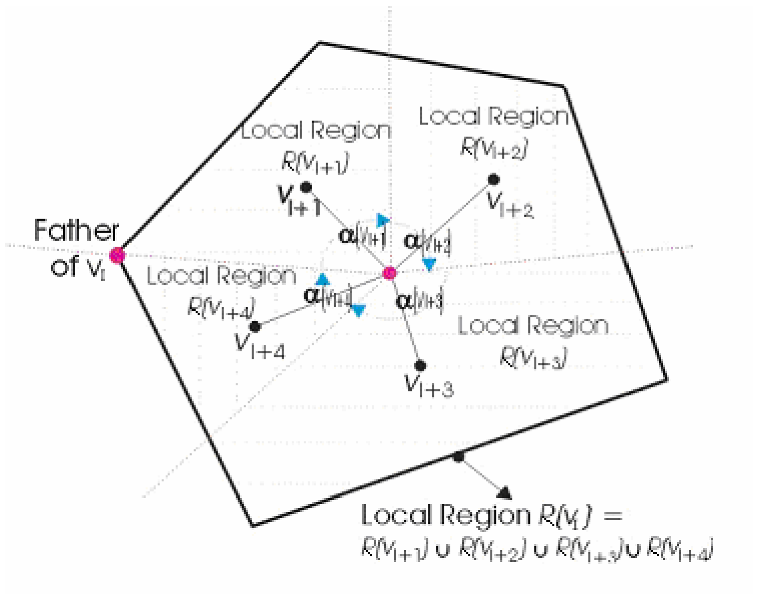

Father of v_i
Local Region $R(v_{i+1})$
Local Region $R(v_{i+2})$
Local Region $R(v_{i+3})$
Local Region $R(v_{i+4})$
v_{i+1}
v_{i+2}
v_{i+3}
v_{i+4}
Local Region $R(v_i) = R(v_{i+1}) \cup R(v_{i+2}) \cup R(v_{i+3}) \cup R(v_{i+4})$

Figure 2.5. An example of partitioning v_i' s local region into four sub-regions.

2.3.2 Local Area Partition – Approach 2

This section presents a variation of the local area partition's algorithm for *space-optimised trees*. Similar to the first approach, this technique also partitions the local areas of vertices using on wedges. However, a wedge, in this approach, is now then redefined so that the areas of local regions are dependant to the weight of vertices. This approach aims to produce better layout in terms of space optimisation, consistency of the layout and the balance of visualisation. Nevertheless, this technique is slightly more computational-expensive compared to the original algorithm. Detail of this approach is described below.

The new partitioning algorithm follows the same process as the approach 1. The weight *w(v)* of a vertex *v* is calculated with the same formula (see Equation 2.1), but the constant C is assigned to 0.45 instead of 0.6 in this approach. The second approach also uses a wedge to define the local region and position of a vertex. The calculation of the *local region* can be found at section 2.3.1.2. As from section 2.3.1, the first layout algorithm divides the boundary polygon by using wedges where the wedges' angles are dependent on the weight of vertices. Thus, the actual areas of polygonal local regions might not be proportional to the wedges' angles (or vertices' weights) because of the different property between an angle and style of a polygon. This approach, however, partitions the boundary polygon into sub-polygons where the areas of these sub-polygons are dependent on the weight of vertices.

Suppose that $A(v_i)$ is the area of local region $R(v_i)$ at vertex v_i, and v has k children $\{v_{l+1}, v_{l+2}, ..., v_{l+k}\}$ and their wedges are respectively $\{wg(v_{l+1}), wg(v_{l+2}), ..., wg(v_{l+k})\}$. The local regions of the child vertices $\{R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k})\}$ are respectively polygons that are formed by the intersection of $\{wg(v_{l+1}), wg(v_{l+2}), ..., wg(v_{l+k})\}$ and $R(v), R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k-1})$ have areas of $\{A(v_{l+1}), A(v_{l+2}), ..., A(v_{l+k})\}$ respectively. The value of $A(v_{l+m})$ of the mth child is calculated by the formula:

$$A(v_{l+m}) = A(v_i)\frac{w(v_{l+m})}{\sum_{j=1}^{k} w(v_{l+j})}$$

Equation 2.3

The area $A(v_{l+m})$ of the local region $R(v_{l+m})$ is dependent on the area magnitudes $A(v_i)$ of its parent's local region as well as the weight of vertex v_{l+m}. In other words, a wedge does not use its angle α to define the size of a local region, but this size is calculated based on the weight ratio of the vertex and its siblings. This algorithm ensures that the local region's area of a vertex is proportional to its weight.

Examples of layouts produced from both approaches on several datasets are presented at section 2.3.6.

2.3.3 Vertex Positioning

This section describes the technical detail of how to define the vertex position inside its polygonal local area. The position of vertex v_i is computed after the calculation of local region. Suppose that the local region $R(v_{l+m})$ of the mth child vertex of v_i is already defined. The next step will calculate the position of the child vertex v_{l+m}.

Firstly, the algorithm finds a point Q in the polygon $R(v_{l+m})$ of vertex v_{l+m} that the straight line connecting Q and its father vertex v_i divides the polygon $R(v_{l+m})$ into two halves of the same area. Next step is to define the position of vertex v_{l+m} from point Q. This position can be anywhere on the segment from the father vertex v_i to point Q and it can be adjusted in order to optimise the visualisation for each type of applications. We, however, decide to position the vertex v_{l+m} simply at the midpoint of the segment in the prototype demonstration (see Figure 2.6). In the special case, when vertex v_i has only one child v_{l+1}, this means its local region and the child's local region are the same. The system will define the position of the child vertex v_{l+1} in the segment from v_i to point Q. In this case, v_i is not a vertex of the local region polygon but a point inside the polygon. The detailed process of finding point Q is described in following paragraphs.

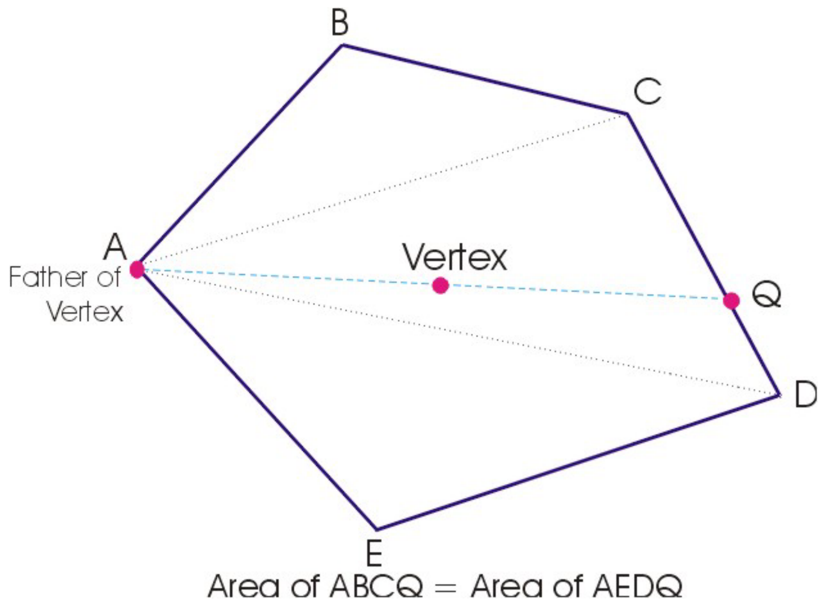


Figure 2.6. An example of positioning a vertex in its local region.

Suppose that R is a local region polygon that has n vertices $\{A_1, A_2, ..., A_n\}$, we need to find a point Q on a polygon R that the segment A_1Q divides R into two small polygons of equal area. There are several possible ways to calculate the position of point Q. We just describe in detail a solution of finding the position Q.

From the first vertex A_1 of R, we divide the polygon R into several triangles namely $\{A_1A_2A_3, A_1A_3A_4, ..., A_1A_{k-1}A_k, ..., A_1A_{n-1}A_n\}$. The area of polygon R is the sum of total areas of triangles $A_1A_2A_3, A_1A_3A_4, ..., A_1A_{k-1}A_k, ..., A_1A_{n-1}A_n$. The next step is to find the side-edge in R that point Q lies on, called A_jA_{j+1}. This process can be executes linearly via the summation of area of these triangles and finishes when the sum of areas of triangles $A_1A_2A_3, A_1A_3A_4, ..., A_1A_{j-1}A_j$ is less than half of the area of R and sum of areas of triangles $A_1A_2A_3, A_1A_3A_4, ..., A_1A_{j-1}A_j, A_1A_jA_{j+1}$ is greater or equal to half of the area of R. When the side-edge A_jA_{j+1} has been found, the next step is to compute the position of Q on A_jA_{j+1}. This process can be narrowed as finding a point Q on a side A_jA_{j+1} of the triangle $A_1A_jA_{j+1}$ which the areas of triangles A_1A_jQ and $A_1A_jA_{j+1}$ are already known. The position of point Q can be easily calculated using geometrical formulas found in most of popular geometry books. Figure 2.6 shows an example of the positioning of a vertex via point *Q*.

2.3.4 Graphical Properties for Displaying Space-Optimised Tree

In order to display the complete hierarchical structure of space-optimised tree thousands of vertices and edges in a limited display space, we assign different node sizes to vertices at the different levels, and also assign different link thicknesses to edges at the different levels of the hierarchy. Particularly the closer to the root, the larger the node size and the wider of edges are. It is quite possible in our visualisation that some nodes and edges might be graphically invisible if they are in the lower levels of the hierarchy. This improves the clarity of the presentation of hierarchical structure by avoiding overcrowded and overlaps among graphical nodes and links. We then display the visualisation on the screen. These graphical properties can be adjusted via an interactive menu to suit with user's preference of viewing.

2.3.5 Computational Complexity of the Algorithm

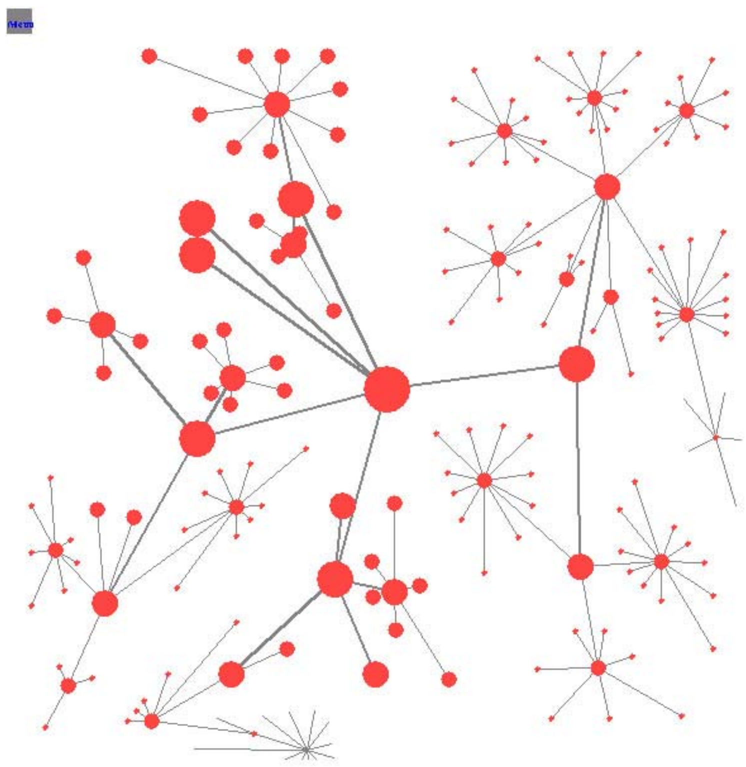
The computational time for running the above two layout algorithms (approach 1 and approach 2) involves the cost of three separate processes including calculating the weights of all vertices, partitioning the local regions of all vertices, and positioning all vertices on the geometrical area. The process of calculating the weights of all vertices involves a post-order traversal through the tree. This means that all nodes are visited only once so the running time for this step is linear. The other two processes, computing the bounded polygons and the location of all vertices, are mixed in sequence. In other words, the layout algorithm partitions the local region of a vertex, and then locates the position of this vertex inside the local region; and so on for other vertices. This composite process involves an in-order traversal through the tree which is also linear. The sum of these linear processes is also linear. As a result, the total computational complexity of the *space-optimised tree* layout algorithm is linear or O(N).

2.3.6 Examples of the Space-Optimised Tree's Visualisation

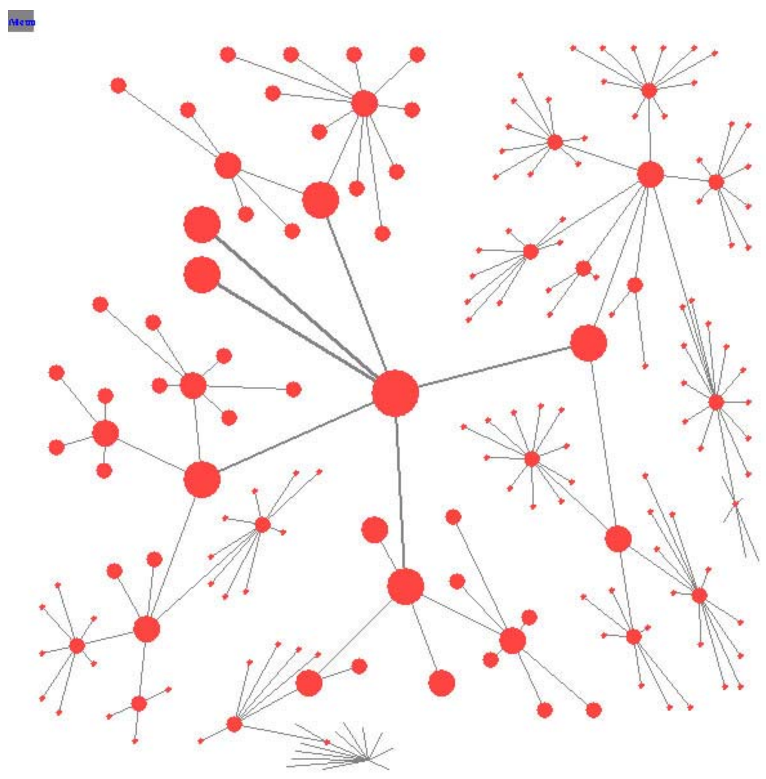
Figure 2.7 through Figure 2.10 show examples of applying the *space-optimised tree* layout algorithm on various datasets. These figures illustrate the layouts generated by both approaches in which images labelled (a) show the layout generated by approach 1 and images labelled (b) show the layouts generated by approach 2. These figures are screen dumps collected from a personal computer with the hardware specification of 800 Mhz CPU, 256 Mbs RAM, 17 inches monitor at 1024x768 resolution. The sizes of the figures' display spaces are all 750x750 pixels.

- Figure 2.7 shows the visualisation of a medium large size dataset of approximately 170 nodes where viewers can easily see all data items and relations among nodes. The running times of the figures in both approaches are less than a second.
- Figure 2.8 shows the visualisation of an entire file system of approximately 3,700 directories and files. Note that the positions of vertices in this figure are slightly modified so that the leaf vertices locate two-third in the segments instead of at the middle (see section 2.3.3). This figure shows clearly that three directories in this file system contain many more sub-directories and files than others. The typical running time of the Figure 2.8 is approximately 6 seconds.
- Figure 2.9 shows the visualisation of a very large uniform data structure of approximately 22,000 nodes. This figure presents well the uniform property of the dataset which each non-leaf node has four children. The typical running time of the Figure 2.9 is approximately 2 minutes and 45 seconds.
- Figure 2.10 shows the visualisation of a huge relational dataset of approximately 50,000 nodes. Overall, the amount of data does not significantly affect the display of the overall tree structure by using our layout technique. The typical running time of the Figure 2.10 is approximately 7 minutes.

These examples illustrate the high capability of our technique for visualising large hierarchical structures for various datasets.

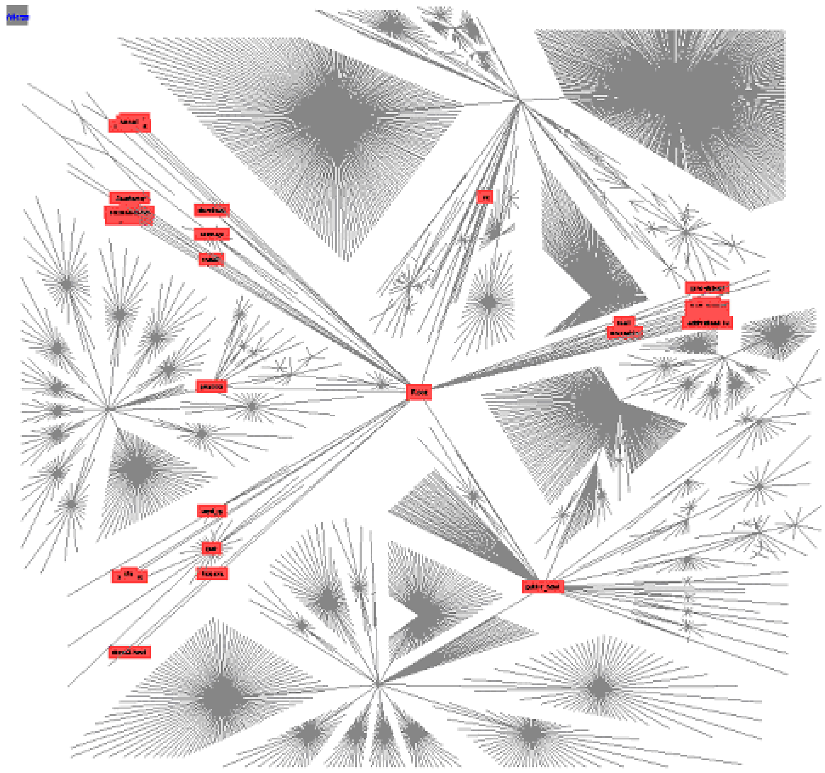


(a)

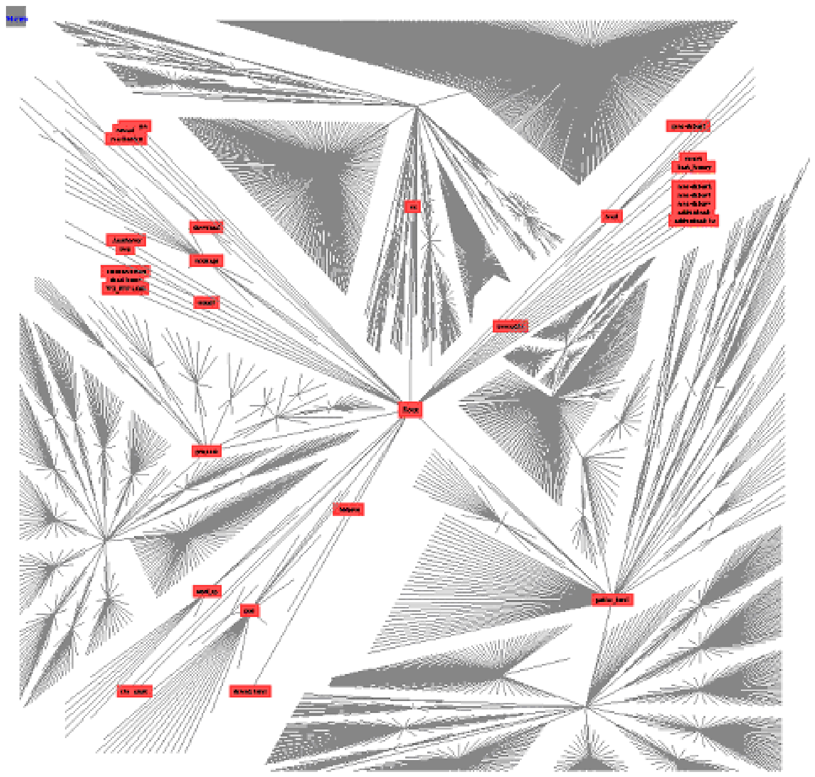


(b)

Figure 2.7. An example of a medium large dataset of approximately 170 nodes. That uses algorithms from respectively (a) approach 1 and (b) approach 2.

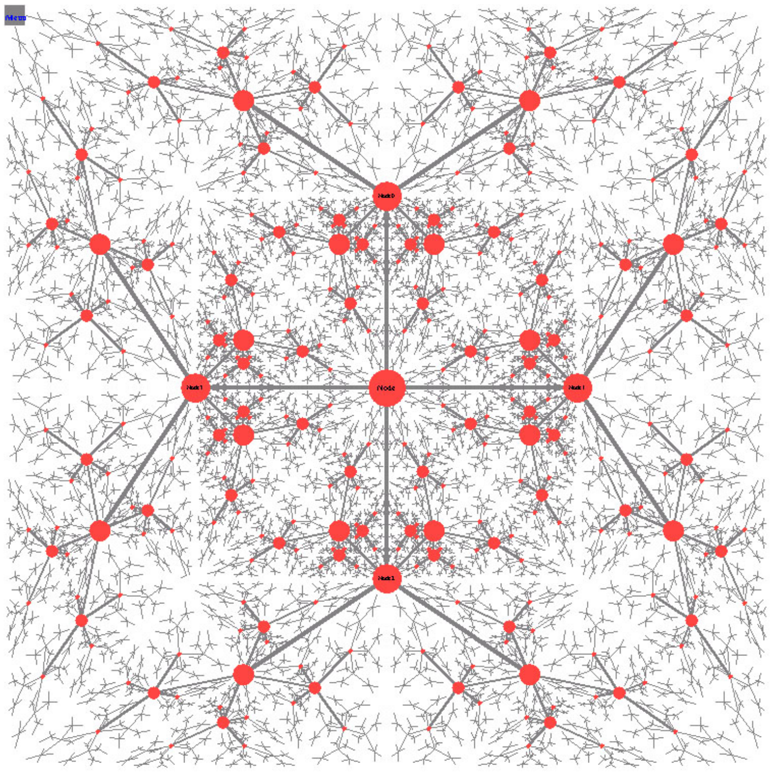


(a)

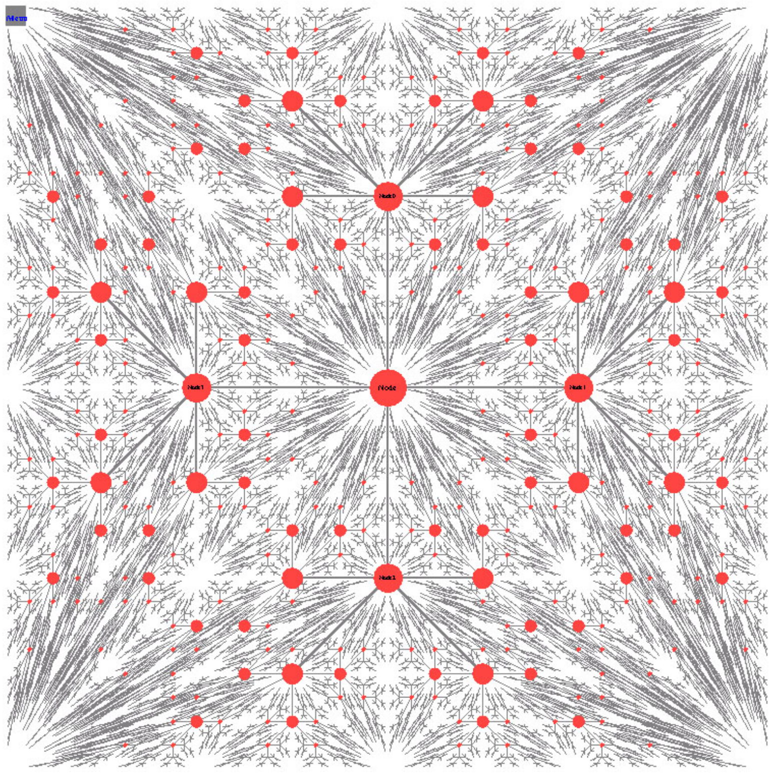


(b)

Figure 2.8. An example of a file system with approximately 3,700 nodes. That uses algorithms from respectively (a) approach 1 and (b) approach 2.

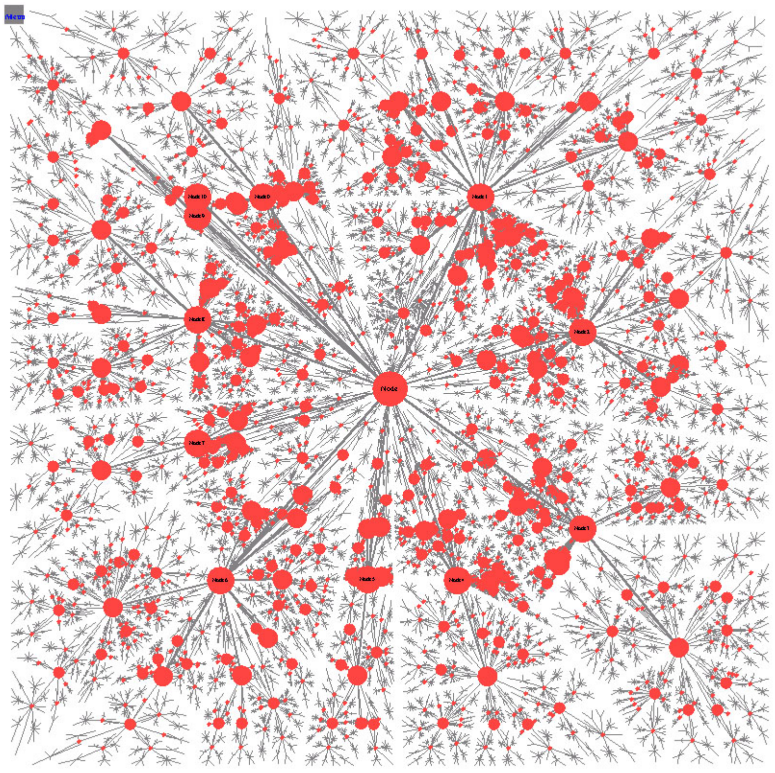


(a)



(b)

Figure 2.9. An example of a uniform dataset of approximately 22,000 nodes. That uses algorithms from respectively (a) approach 1 and (b) approach 2.


(a)

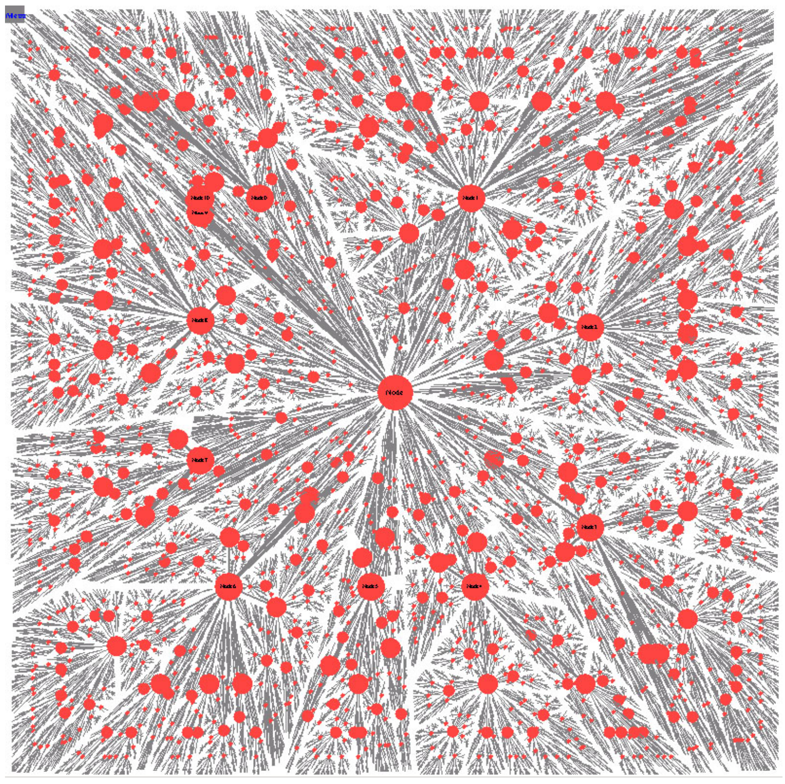

(b)

Figure 2.10. An example of a huge dataset of approximately 50,000 nodes. That uses algorithms from respectively (a) approach 1 and (b) approach 2.

2.4 EncCon Tree

EncCon tree is an alternative *enclosure+connection* approach for visualising and navigating large hierarchical information in a two-dimensional space. *EncCon tree* layout algorithm inherits the advantage of *space-optimised tree* or *SO-Tree* (see section 2.3) that maximises the utilisation of display space by using connection approach. However, this technique uses a rectangular division method for partitioning the layout, rather than a polygonal partitioning method used in the *SO-Tree*. This partitioning concept aims to provide users a more straightforward way to perceive the relational information. The detailed technical specification of the *EncCon tree* is now further described.

EncCon tree layout algorithm is responsible for the positioning of a set of vertices $\{v_1, v_2, ..., v_n\}$ of a tree T and the computation of rectangular local regions $\{R(v_1), R(v_2), ..., R(v_n)\}$ for the drawing of sub-trees $\{T(v_1), T(v_2), ..., T(v_n)\}$ in a two-dimensional plane. Similarly to the *space-optimised tree*, the *EncCon tree* also uses enclosure approach to partition the geometrical display which each vertex v_i is bounded by a rectangular local region $R(v_i)$. The area of $R(v_i)$ is equal to the sum of the areas of its children. The drawing of a sub-tree $T(v_i)$ is restricted within the boundary of the rectangle $R(v_i)$ (see an example in Figure 2.2). A geometrical location of sub-tree $T(v_i)$ rooted at v_i is calculated based on the weight $w(v_i)$ of v_i and its local region $R(v_i)$. The local regions produced by the *EncCon tree* are square-like rectangles.

Squarified tree-maps (Bruls, Huizing & van Wijk 2000) also aims to produce square-like rectangles and it produces similar space partitioning outcomes as the *EncCon tree* does. Therefore, the partitioning outcomes produced by the *squarified tree-maps* also satisfy the space partitioning requirements of the *EncCon tree* visualisation. However, technically these two approaches are different. The *EncCon tree* partitioning algorithm places the local regions around four sides (east-south-west-north) of the parent rectangle in a circular manner (see Figure 2.11a), while most of other space-filling algorithms including the *squarified tree-maps* partition rectangles in a vertical-horizontal manner as illustrated in Figure 2.11b. This partitioning algorithm ensures the efficiency of space utilisation as most of the space-filling approaches do. This process also aims to enhance a good distribution of nodes and edges of a given node-link diagram in its rectangular local region.

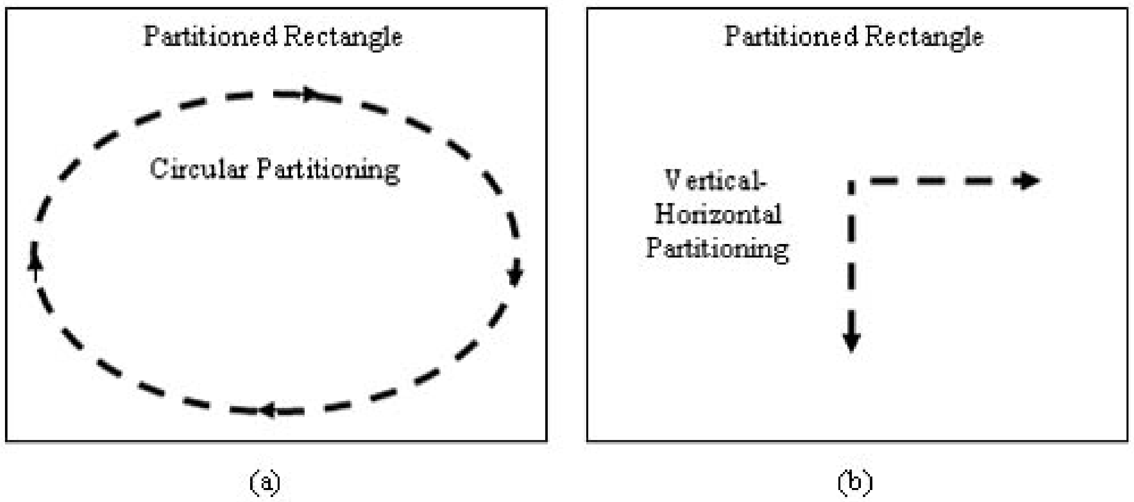


Figure 2.11. Diagram of partitioning direction.
Diagram (a) shows the partitioning direction in *EncCon tree* and diagram (b) shows the partitioning direction in *squarified tree-maps*.

The geometrical position of a sub-tree $T(v_i)$ rooted at vertex v_i is calculated based on the properties of v_i and its local region $R(v_i)$. Each vertex is associated with a weight and the size of vertex's local region is calculated proportionally to its weight. The process of weight calculation is independent to layout algorithm and is executed before the geometrical partitioning. Algorithm for calculating vertices' weights is described at section 2.1. The partitioning algorithm is firstly illustrated from an example presented in the next subsection.

2.4.1 An Example of the Partition

Suppose we have a rectangle R with width = 6 and height = 4, we need to divide this rectangle into 5 sub-rectangles $\{R_1, R_2, R_3, R_4, R_5\}$ whose weights are respectively {4, 4, 2, 1, 2}. The starting partitioning side is assumed at the left-hand side. The weight of this rectangle is 13 which equals to the sum of the weights from the above sub-rectangles.

The first step is to add a single rectangle with weight = 4 into the first partitioning side, i.e. left-hand side. This step produces a rectangle R_1 whose width and height are respectively $Width(R_1) = 4*6/13 = 1.85$ and $Height(R_1) = 4$. Next, we add the second rectangle R_2 (weight = 4) below the first rectangle R_1, i.e. they both share the common left-hand side from rectangle R. These two rectangles will have the same dimension respectively of *Width(R*$_1$*) = 8*6/13 = 3.69*, *Height(R*$_2$*) = 4*4/8 = 2*, and *Width(R*$_2$*) = 8*6/13 = 3.69*, *Height(R*$_2$*) = 4*4/8 = 2*. The third step tries to insert the third rectangle *R*$_3$ (weight = 2) below rectangle *R*$_2$. However, this process is dismissed because the rectangle *R*$_3$ is too thin whose width and height are respectively *Width*(*R*$_3$) = *10*6/13 = 4.62* and *Height(R*$_3$*) = 2*4/10 = 0.8*. We now start the second partitioning circle, which the partitioning side moves from the left-hand side to the top side. The *remaining rectangle* for the partitioning now has width and height of 2.31 and 4. These steps are repeated on other sides of the *remaining rectangle* until all sub-rectangles *{R*$_1$*, R*$_2$*, R*$_3$*, R*$_4$*, R*$_5$*}* are placed inside the large rectangle *R* (see Figure 2.12). We next describe the technical details of the partitioning algorithm.

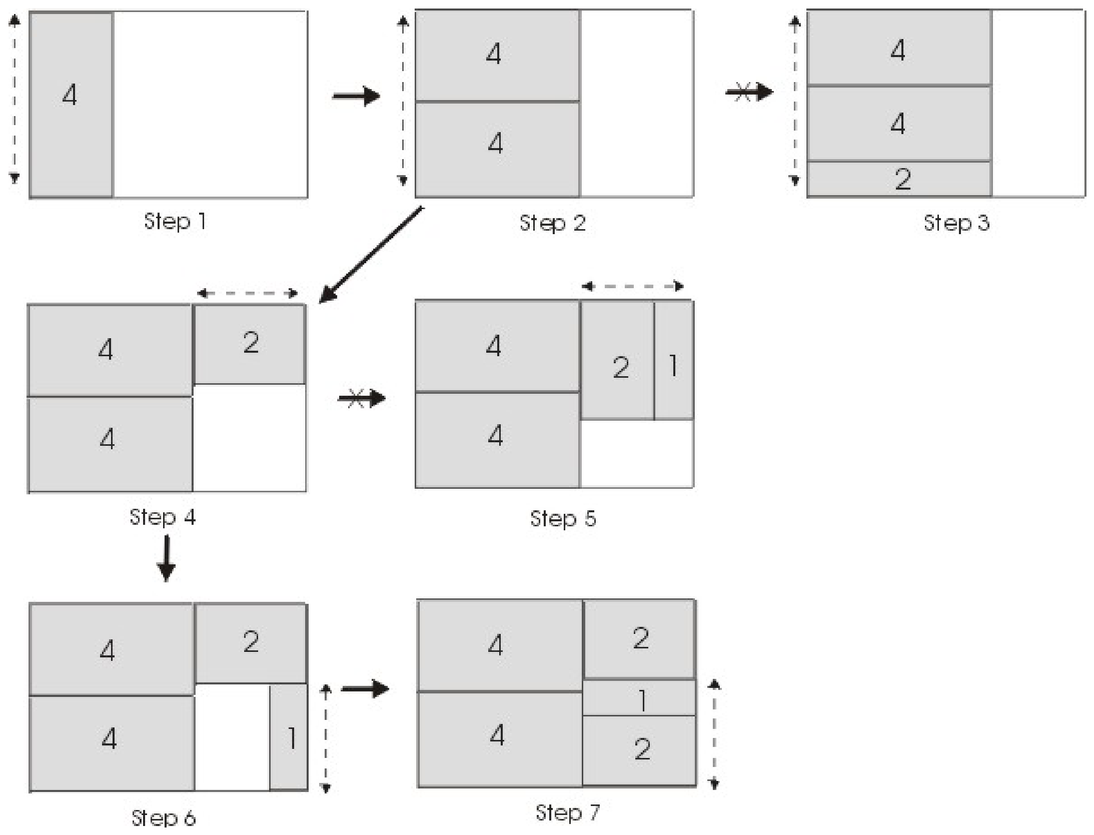


Figure 2.12. An illustrated example of the partitioning process.

2.4.2 Local Region Partition

Similarly to *space-optimised tree*, this algorithm firstly defines the local region *R(v*$_r$*)* for root vertex v_r as the entire rectangular display area, and then the root v_r is positioned at the centre of *R(v*$_r$*)*. Then, all other local regions *{R(v*$_1$*), R(v*$_2$*), ..., R(v*$_n$*)}* are calculated in sequence from the root outward to the leaf vertices. The sequence of this partitioning process is also described at section 2.3.

Suppose that the rectangular local region $R(v_i)$ of vertex v_i is already defined, and the position of vertex v_i is at the centre of $R(v_i)$. We then need to calculate the local regions $\{R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k})\}$ for all the child vertices $\{v_{l+1}, v_{l+2}, ..., v_{l+k}\}$ of v_i. This partitioning ensures that the size of rectangle $R(v_{l+i})$ of vertex v_{l+i} is proportional to its weight $w(v_{l+i})$. The further description of the partitioning of a rectangle $R(v_i)$ into sub-rectangles $\{R(v_{l+1}), R(v_{l+2}), ..., R(v_{l+k})\}$ is as below.

Assumably, the vertex μ is the parent of vertex v_i. We firstly find a side on rectangle $R(v_i)$, called *initial-side*, where vector $\overrightarrow{\mu v_i}$ cuts the rectangle $R(v_i)$ or where it is closest to the vertex μ if vector $\overrightarrow{\mu v_i}$ does not cut the rectangle. If v_i is the root vertex then its *initial-side* is defaulted as the bottom-side of $R(v_i)$. When the *initial-side* of rectangle $R(v_i)$ is defined, the algorithm starts it partitioning. This process starts on the opposite side of the defined *initial-side*, and the partitioning is applied respectively to all four sides of $R(v_i)$ in a clock-wise direction. This procedure is shown in Pseudo Code 2.1:

```
procedure rectangle-partition(array children, rectangle div-rect)
begin
  rectangle remaining-rect := firstside-partition(children, div-rect);
  if remaining-rect != null then
    remaining-rect := secondside-partition(get_remaining_chidlren(), remaining-rect);
    if remaining-rect != null then
      remaining-rect := thirdside-partition(get_remaining_chidlren(), remaining-rect);
      if remaining-rect != null then
        remaining-rect := fourthside-partition(get_remaining_chidlren(), remaining-rect);
      fi
    fi
  fi
  return remaining-rect;
end
```

Pseudo Code 2.1

On each side of rectangle *R(v$_i$)*, the partitioning creates and fills in m (m ≤ k) sub-rectangles *{R(v$_{l+s+1}$), R(v$_{l+s+2}$), ..., R(v$_{l+s+m}$)}* with the same width (or height depending on which side). This property is formulated as:

$$Width(R(v_{l+s+1})) = Width(R(v_{l+s+2})) = ... = Width(R(v_{l+s+m})) \text{ or}$$

$$Height(R(v_{l+s+1})) = Height(R(v_{l+s+2})) = ... = Height(R(v_{l+s+m}))$$

Equation 2.4

These sub-regions will form a larger rectangle inside *R(v$_i$),* and thus also leave a rectangular non-partitioned area (see Figure 2.13). This *non-partitioned rectangle* (or also called *remaining rectangle RR(v$_i$)*) will be reused for the next partitioning at the next side of *R(v$_i$)*. This means that for the partitioning at the next side, *R(v$_i$)* is now considered as *RR(v$_i$),* and before the partitioning *RR(v$_i$) = R(v$_i$)*. The value *m* is determined by the size of the *R(v$_i$)* (or *RR(v$_i$)*) as well as the constraint of smallest ratio λ_{l+s+j} (*1 ≤ j ≤ m*) of all ratios *{λ_{l+s+1}, λ_{k+s+2}, ..., λ_{k+s+m}}*, where we have:

$$\lambda_{k+s+j} = \frac{Width(R(v_{k+s+j}))}{Height(R(v_{k+s+j}))}$$

Equation 2.5

Suppose that the partitioning is at a side of the *remaining rectangle*. The algorithm firstly checks the numbers of vertexes that can be added into this side. This aims to maximise the value *m*, but also ensure that every sub-rectangle at the partitioned side is roughly square, which must satisfy the Equation 2.6:

$$\frac{1}{\rho} < \lambda_{k+s+j} < \rho$$

Equation 2.6

where ρ is a constant and $\rho = 1.5$ in our prototype implementation. The checking process to find out the smallest ratio λ_{l+s+j} is quite computationally expensive. In order to reduce this computational cost, the layout algorithm only checks the ratio λ_{l+s+m} of the last child's rectangular local region. Since all the vertices are initially sorted in ascendant order of weight, the ratio λ_{l+s+m} is also the smallest ratio of all $\{\lambda_{l+s+1}, \lambda_{k+s+2}, \ldots, \lambda_{k+s+m}\}$.

As above, there are m child vertices $\{v_{l+s+1}, v_{l+s+2}, \ldots, v_{l+s+m}\}$ of vertex v_i which need to be added into a side of the remaining rectangle $RR(v_i)$ inside the rectangular local region $R(v_i)$, with respective sub-regions $\{R(v_{l+s+1}), R(v_{l+s+2}), \ldots, R(v_{l+s+m})\}$. Suppose that the *width-side* of sub-rectangles $\{R(v_{l+s+1}), R(v_{l+s+2}), \ldots, R(v_{l+s+m})\}$ is same as the direction of the current partitioned side in $RR(v_i)$; the length of partitioned side in $RR(v_i)$ is l_1 and the length of the other side is l_2. Then the width and height of a rectangle $R(v_{l+s+j})$ are calculated respectively by the following formulas:

$$Width(R(v_{l+s+j})) = l_1 \frac{w(v_{l+s+j})}{\sum_{n=1}^{m} w(v_{l+s+n})}$$

Equation 2.7

$$Height(R(v_{l+s+j})) = l_2 \frac{\sum_{n=1}^{m} w(v_{l+s+n})}{w(RR(v_i))}$$

Equation 2.8

Where $w(v)$ is the weight of vertex v, and thus $w(v_{k+s+j})$ is the weight of vertex v_{k+s+j} which is already calculated from section 2.1. $w(RR(v_i))$ is the weight of the *remaining rectangle* $RR(v_i)$. $w(RR(v_i))$ is initially defined as the sum of total weight of all the children $\{v_{l+1}, v_{l+2}, \ldots, v_{l+k}\}$ which is also the weight of rectangle $R(v_i)$. The value of $w(RR(v_i))$ after this partitioning is calculated by formula:

$$w(RR(v_i)) = w(RR(v_i)) - \sum_{n=1}^{m} w(v_{l+s+n})$$

Equation 2.9

The above formulas ensure the size of rectangular local area of each vertex is proportional to its weight or the numbers of descendants. This partitioning repeats around 4 sides of the rectangle $R(v_i)$. Figure 2.13 illustrates an example of the partitioning on the left side of a remaining rectangle.

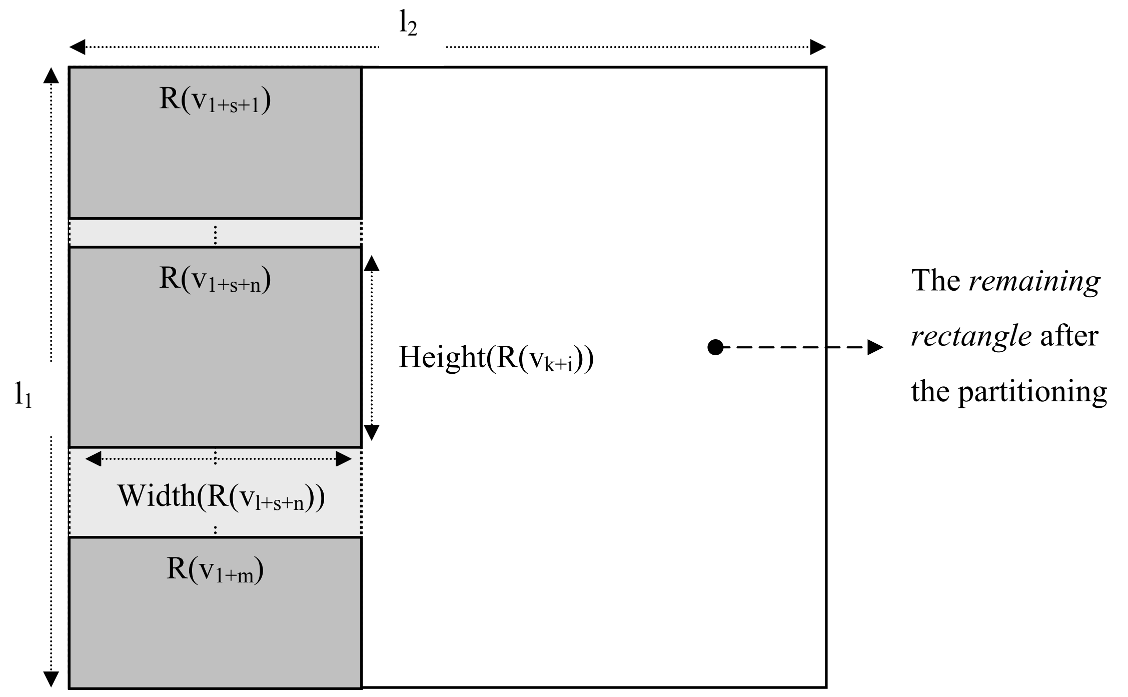


Figure 2.13. An example of the partitioning on the left side of the rectangle.

However, this division might not complete because the ratio of width and height of a particular rectangle from Equation 2.5 does not satisfy the constraint of Equation 2.6. This means that after the partitioning (all around 4 sides), there are still a number of non-partitioned child vertices and the *remaining rectangle* is not empty. This thesis proposes two approaches to solve this problem.

2.4.2.1 Local Region Partition - Approach 1

The initial weight $w(R(v_i))$ of the rectangle $R(v_i)$ increases slightly, so that the total weight of all children $\{v_{l+1}, v_{l+2}, ..., v_{l+k}\}$ is less than $w(R(v_i))$. This also means, initially the weight of *remaining rectangle* $RR(v_i)$ (also initially is $R(v_i)$) is greater than the sum of weights of all child vertices. Then, the partitioning around four sides of the rectangle are reprocessed (see Pseudo Code 2.1). If this partitioning still does not complete because of the constraint of Equation 2.6, the initial weight $w(R(v_i))$ continues to increase and all the above steps are repeated until all the sub-rectangles are fitted inside the rectangle $R(v_i)$. The above steps ensure all the sub-rectangles are placed circularly around four sides of the $R(v_i)$. Thus, the aesthetics of the overall enclosure layout is retained. Nevertheless, this approach also looses the utilisation of display space there might be unused space inside the partitioned rectangles. Figure 2.14 shows an example of a layout of a medium-large dataset using this approach. This approach is formally defined as procedure below:

```
**procedure** partition (**array** children, **rectangle** div-rect)
**begin**
  **rectangle** remaining-rect = rectangle-partition(children, div-rect);
  **if** remaining-rect != null **then**
    div-rect.weight := C*div-rect.weight;   /* C is a constant to increase the weight*/
    partition(children, div-rect);
  **fi**
**end**
```

Pseudo Code 2.2

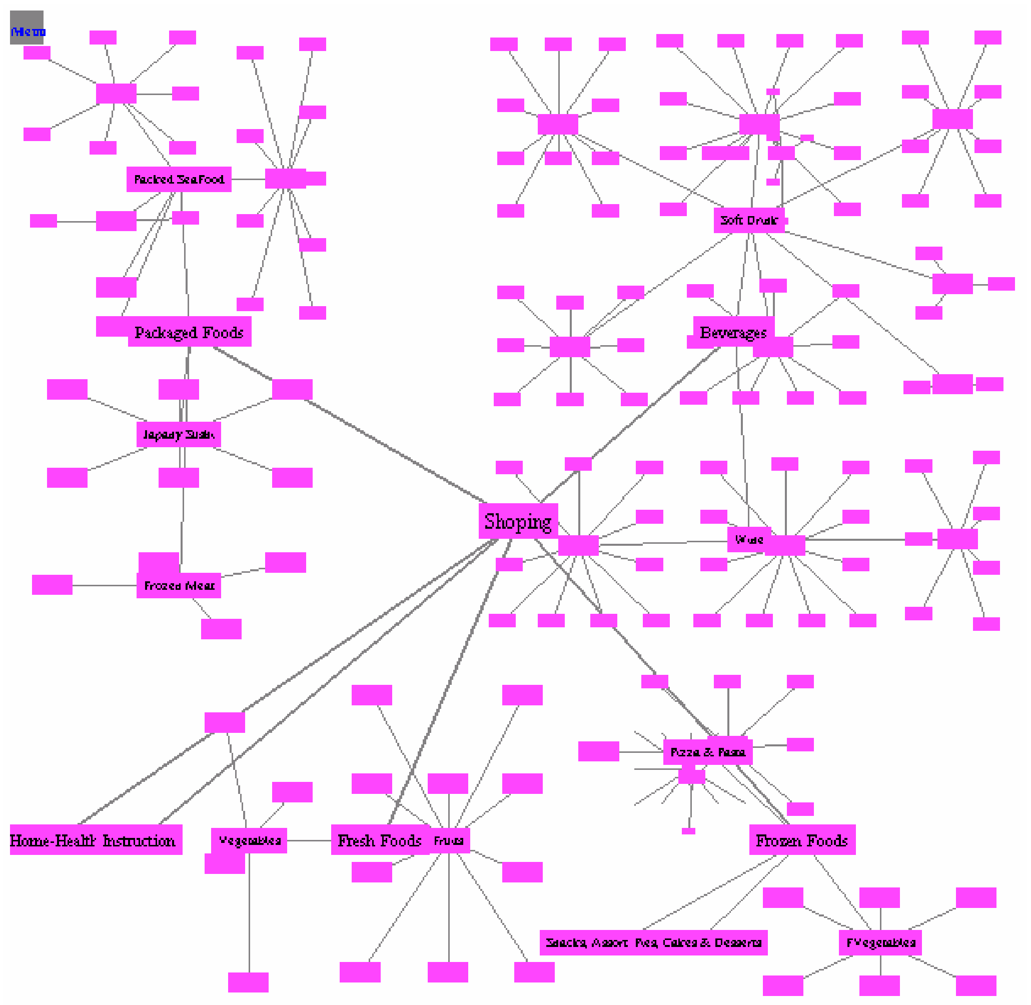


Figure 2.14. An example of *EncCon tree* visualisation using approach 1.

2.4.2.2 Local Region Partition - Approach 2

In this approach, the partitioning continues to execute at the *remaining rectangle* $RR(v_i)$ until all local regions of remaining child vertices are fitted inside the rectangle $R(v_i)$. This means that the partitioning procedure (see Pseudo Code 2.1) is repeated where the partitioned rectangle is now the *remaining rectangle* $RR(v_i)$ and the list of children are remaining vertices. If this step still does not complete, the above procedure is repeated until all the sub-rectangles are fitted inside the rectangle $R(v_i)$. The constraint of Equation 2.6 is not applied to the procedure of partitioning to the last node on the side. This property ensures the partitioning loop is finite.

The above algorithm ensures 100% space efficiency. However, it also does not ensure that all the sub-rectangles are placed circularly around four sides of a rectangle. Thus, it might also reduce the aesthetics of the overall enclosure layout in comparison with the first approach. This approach works best with the list of vertexes in ascendant order of weight. Thus, the list of all vertices $\{v_1, v_2, ..., v_n\}$ is sorted in ascendant order of weights. This step is executed before starting the partitioning of the sub-regions for these vertices. Figure 2.15 shows an example of a layout of a medium-large dataset using approach 2; this example uses the same dataset as in Figure 2.14. The second approach is formally defined as produce below:

```
procedure partition (array children, rectangle div-rect)
begin
  rectangle remaining-rect = rectangle-partition(children, div-rect);
  if remaining-rect != null then
    partition(get_remaining_chidlren(), remaining-rect);
    fi
end
```

Pseudo Code 2.3

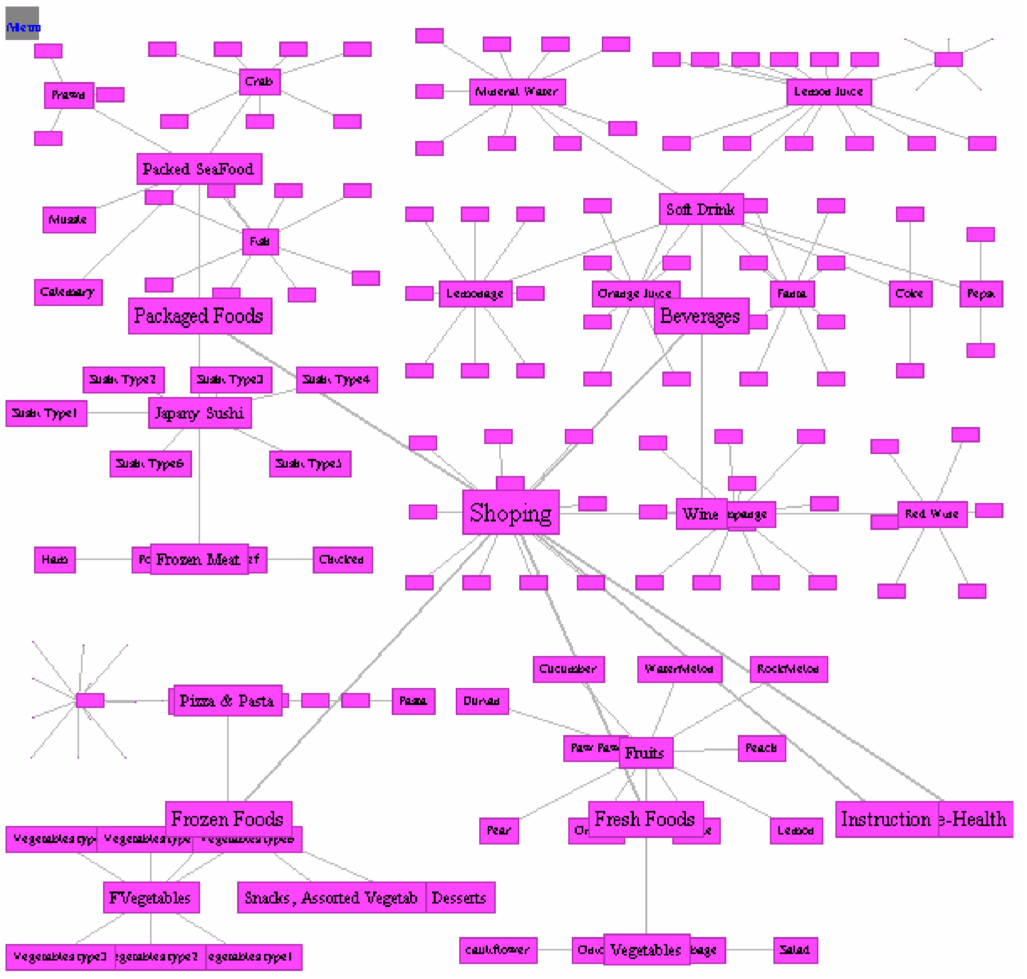


Figure 2.15. An example of *EncCon tree* layout using approach 2.

2.4.3 Graphical Properties for Displaying EncCon Tree

The design rule and the assignments of graphical properties to nodes and links in *EncCon tree* is similar to those we used in *space-optimised tree* (see section 2.3.4). Therefore, we skip the full description of this step.

2.4.4 Computational Complexity of the Algorithm

The computational time for running the above *EncCon tree* layout algorithms (approach 1 and approach 2) involves the cost of three separate processes including the weight calculation of all vertices, nodes sorting, and the local regions partitioning of all vertices. Similarly to *space-optimised tree*, the process of calculating the weights of all vertices involves a post-order traversal through the tree. This means all nodes are visited only once so the running time for this step is linear. The sorting process involves all children of a vertex. We used the quick sort algorithm which has the complexity of Nlog(N). This sorting process is independent to the partitioning algorithm and it runs before the partitioning step. The partitioning process uses an in-order traversal through the tree so that the local region of a node is defined before partitioning the local regions for its children. In detail, at each node, the layout algorithm calculates the local regions of all its children around the sides of the node's rectangular local region until all the children's local regions are fitted inside the rectangle.

The computational cost for the partitioning at each side involves the adding of m children into the side. This step includes the process of counting the number of child vertices that can be added into the side, the process of calculating the width and height of each local region and finally the weight of the remaining rectangle after the partitioning. The process of counting the number of children has m steps to ensure the ratio of width and height a rectangle satisfies Equation 2.6. Each step uses a constant processing time because the algorithm only needs to check the ratio of the last child (see section 2.4.2). Thus, total processing time for partitioning m child vertices at the side of the rectangle is mC where C is a constant.

The computational cost of rectangle partitions includes the cost of the partition around four sides of the rectangle (see Pseudo Code 2.1). Suppose that $\{m_1, m_2, m_3, m_4\}$ are respectively the numbers of children that can be added into each side of the rectangle. The layout algorithm only adds the child vertices inside their father's local region, thus we have: $m_1 + m_2 + m_3 + m_4 \leq k$, where k is the total numbers of children of the vertex. Thus, the processing time for partitioning k children into a given rectangle is less than kC. In other words, the worst case for running Pseudo Code 2.1 is kC.

This thesis proposes two approaches for this partitioning whose procedures are described respectively at Pseudo Code 2.2 and Pseudo Code 2.3. The computational complexity of each algorithm is described below.

- The first approach increases the weight of the *remaining-rectangle* and repeats process from Pseudo Code 2.1. This process stops when all the local regions of the child vertices are inserted properly inside the large rectangle. There are finite repetitions of the above processes. Thus, the computational cost in worse case for this approach is rkC, where r is the repeating times for the recalculation from Pseudo Code 2.1 and r is finite. This partitioning involves the in-order traversal through the tree where each node is visited only one. Consequently, the total cost for partitioning the entire tree hierarchy can be considered as *rnC*, where *r* is a finite number indicating the repeating times and *n* is the number of vertices. In other words, this computational cost of this approach is linear or O(N).

- The second approach continues the process from Pseudo Code 2.1 for the remaining rectangle and non-partitioned children and so on. This loop exists when all the local regions of the child vertices are added inside the large rectangle. Thus, the computational cost for this approach is the sum of $\{k_1C + k_2C + \ldots + k_nC\}$, where k_i is number of children that can be added inside a *remaining-rectangle*. Obviously, we know that the sum of $k = \{k_1 + k_2 + \ldots + k_n\}$ is also the numbers of the child vertices. Similarly to approach 1, the partitioning involves the in-order traversal through the tree; therefore, the total cost for partitioning the entire tree hierarchy is as *nC*, where *n* is the number of vertices. In other words, this computational cost of this approach is also linear or O(N).

The sum of weight calculation and partitioning processes are also linear. Although the sorting algorithm is not linear, it is independent to the partitioning process. Therefore, regardless this sorting process, the total computational complexity of the *EncCon tree* layout algorithm is linear or O(N).

2.4.5 Examples of the EncCon Tree

Figure 2.16 through Figure 2.21 are examples of applying the *EncCon tree* layout algorithm on various datasets. From the experiments, approach 2 often produces more efficient and better layouts than approach 1 does. Thus, the algorithm from second approach is used in most of our applications. Figure 2.16 and Figure 2.17 are examples of approach 1. And Figure 2.18 to Figure 2.21 are four examples of approach 2. All these figures are screen dumps collected from a personal computer with the hardware specification of 800 Mhz CPU, 256 Mbs RAM, 17 inches monitor at 1024x768 resolution. The sizes of the figures' display spaces are all 750x750 pixels.

- Figure 2.16 shows the visualisation using approach 1's algorithm of a large file system of approximately 2,750 directories and files. This figure shows clearly the abstract view of the file system, such as the directory 'LiveNet' contains a folder with very large numbers of files (at the near bottom of the figure). The typical running time of this dataset is approximately 5 seconds.

- Figure 2.17 shows the visualisation using approach 1's algorithm of a very large file system of approximately 11,000 directories and files. This figure shows quite well the abstract view of the file system. The typical running time of this figure is approximately 25 seconds.
- Figure 2.18 shows the visualisation using approach 2's algorithm of a medium large dataset of approximately 900 directories and files. This figure displays clearly the structure of the file system, such as the folder 'medijobs' contains one folder in which contains a large numbers of files. The typical running time of this figure is approximately 4 seconds.
- Figure 2.19 shows the visualisation using approach 2's algorithm of a large dataset of approximately 3,660 directories and files. As can be seen from the figure, the directories 'sotree' or 'enccon' or 'focus', etc have mostly three levels where the these directories contain a number of folders and each folder also contains a numbers of files. The typical running time of this figure is approximately 9 seconds.
- Figure 2.20 shows the visualisation using approach 2's algorithm of the entire Java v.1.4.1 Documentation of approximately 9,500 directories and files. This visualisation displays quite clearly the relational structure of information. The typical running time of this figure is approximately 45 seconds.
- Figure 2.21 shows the visualisation using approach 2's algorithm of a huge dataset of approximately 50,000 nodes. The abstract view of the hierarchy is presented well in this visualisation. The average running time of this dataset is approximately 15 minutes.

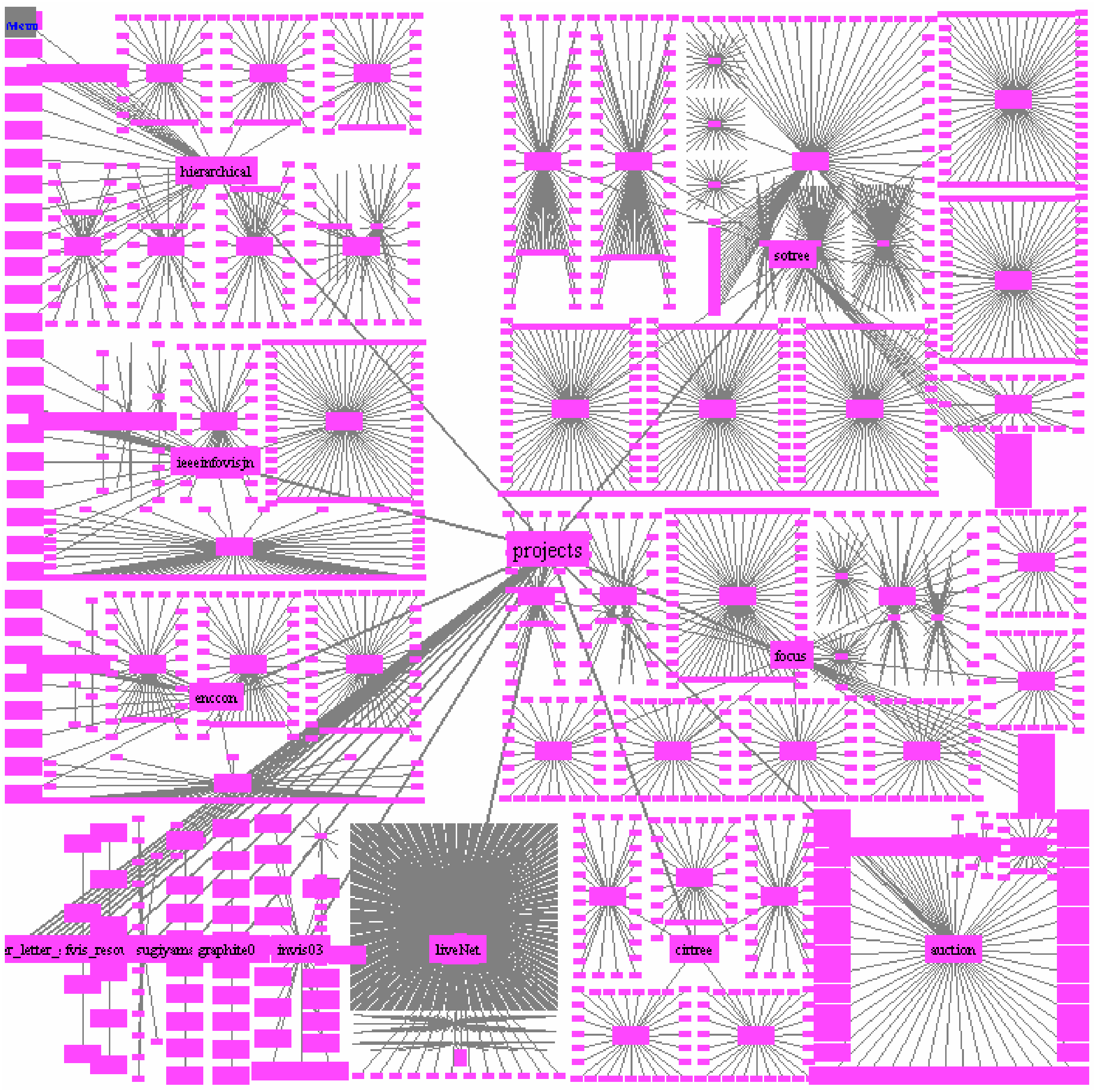


Figure 2.16. An example of a file system with approximately 2,750 nodes.

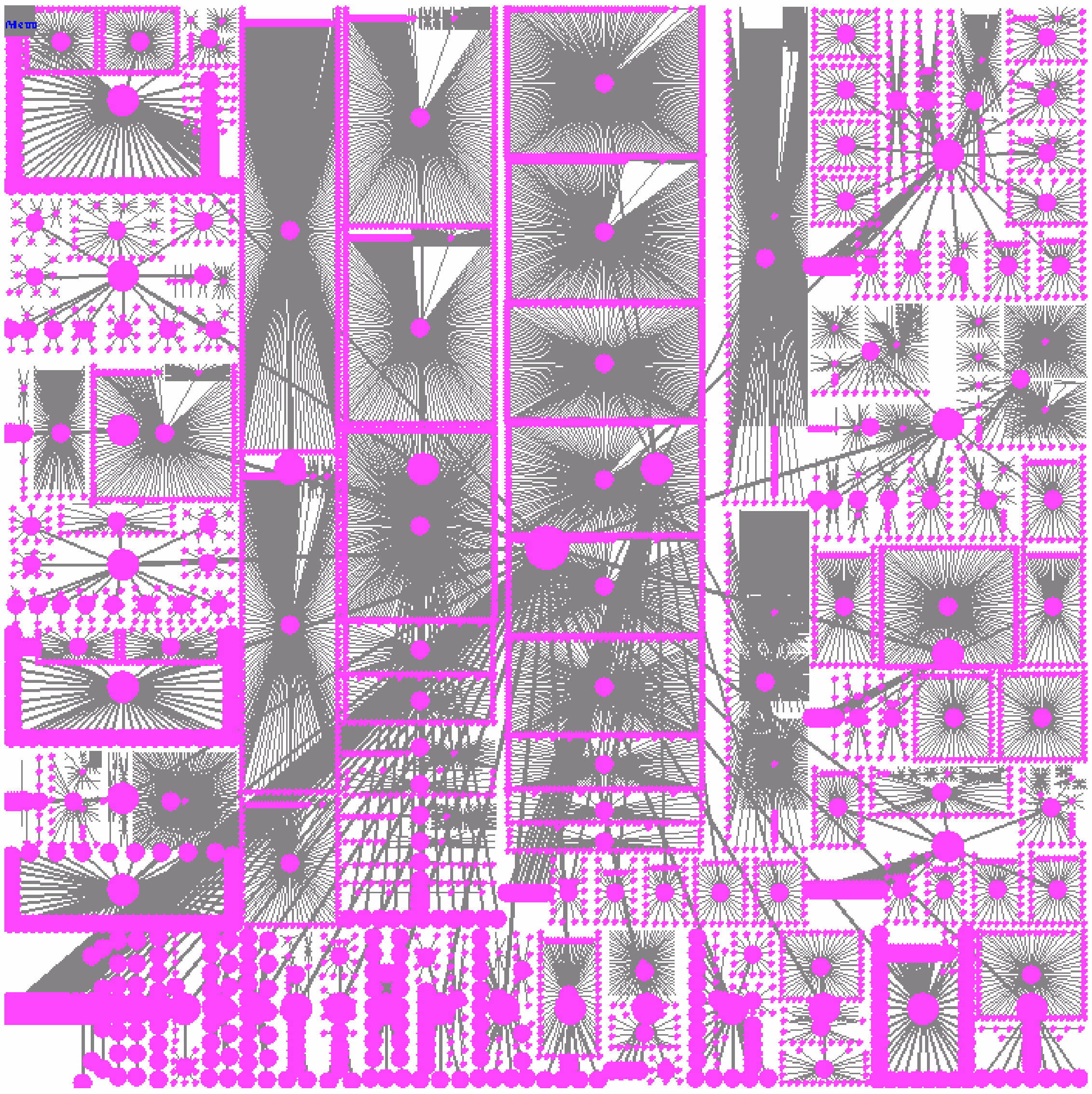


Figure 2.17. An example of a file system with approximately 11,000 nodes.

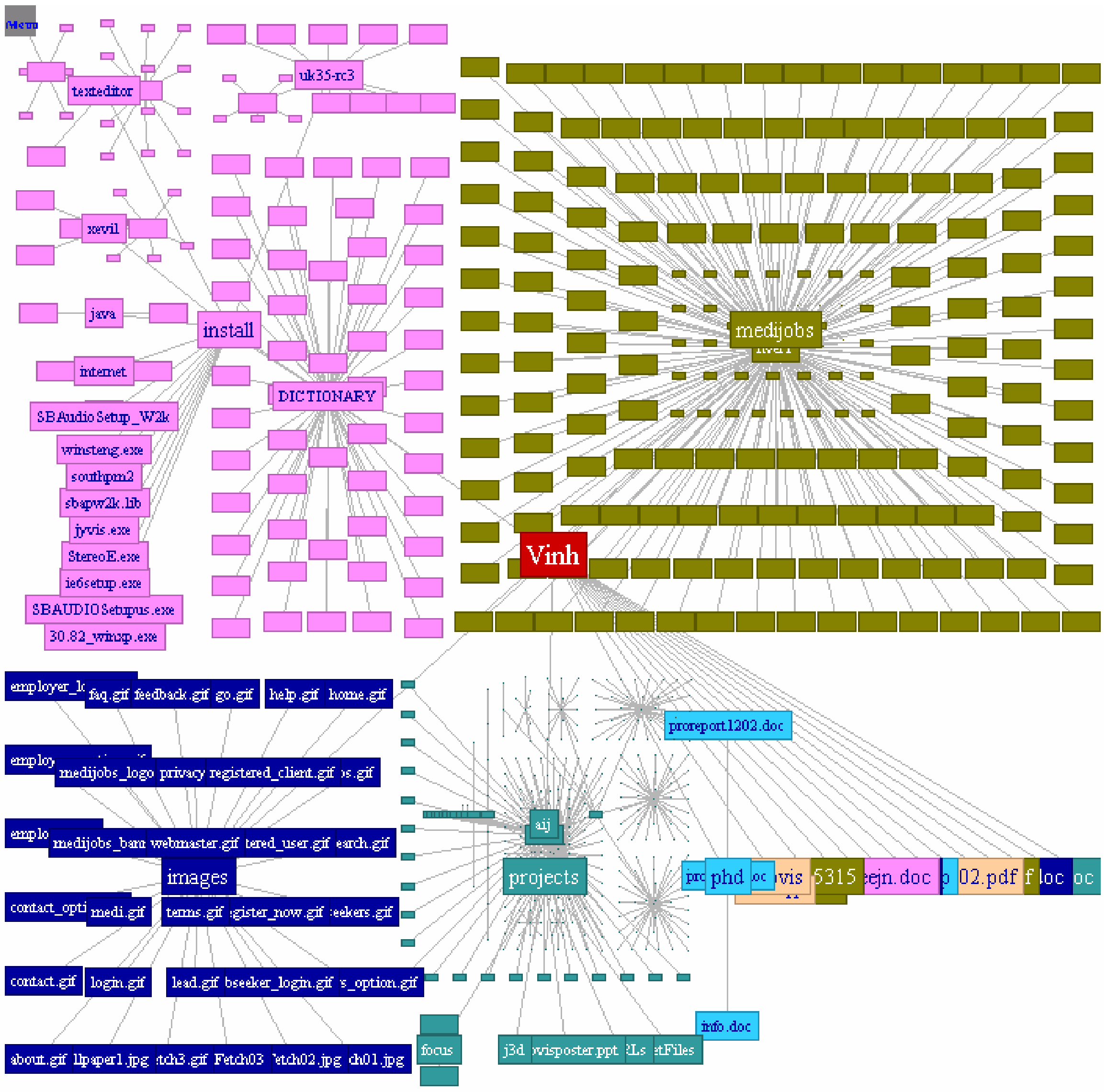


Figure 2.18. An example of a file system with approximately 900 nodes.

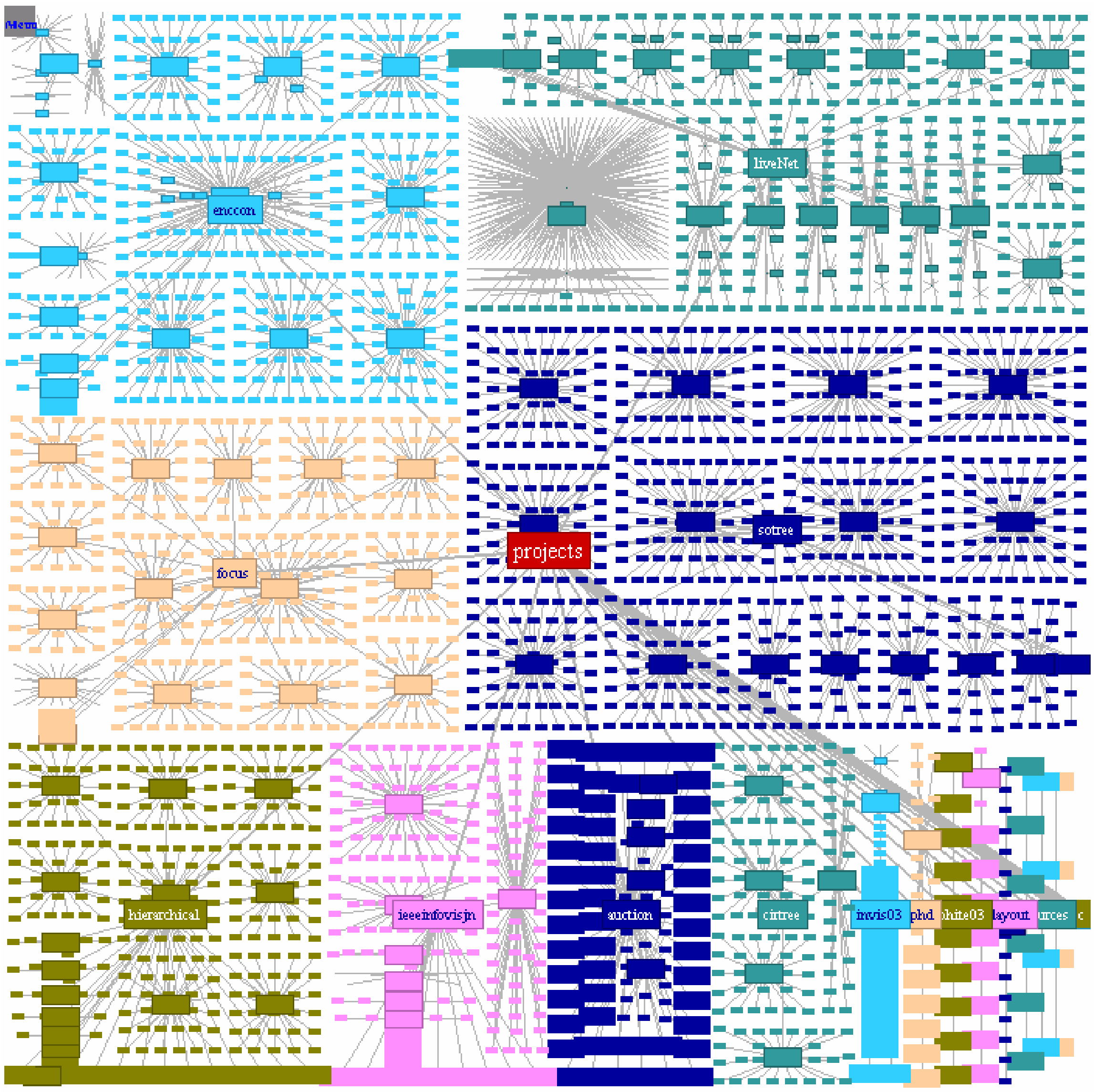


Figure 2.19. An example of a file system with approximately 3,600 nodes.

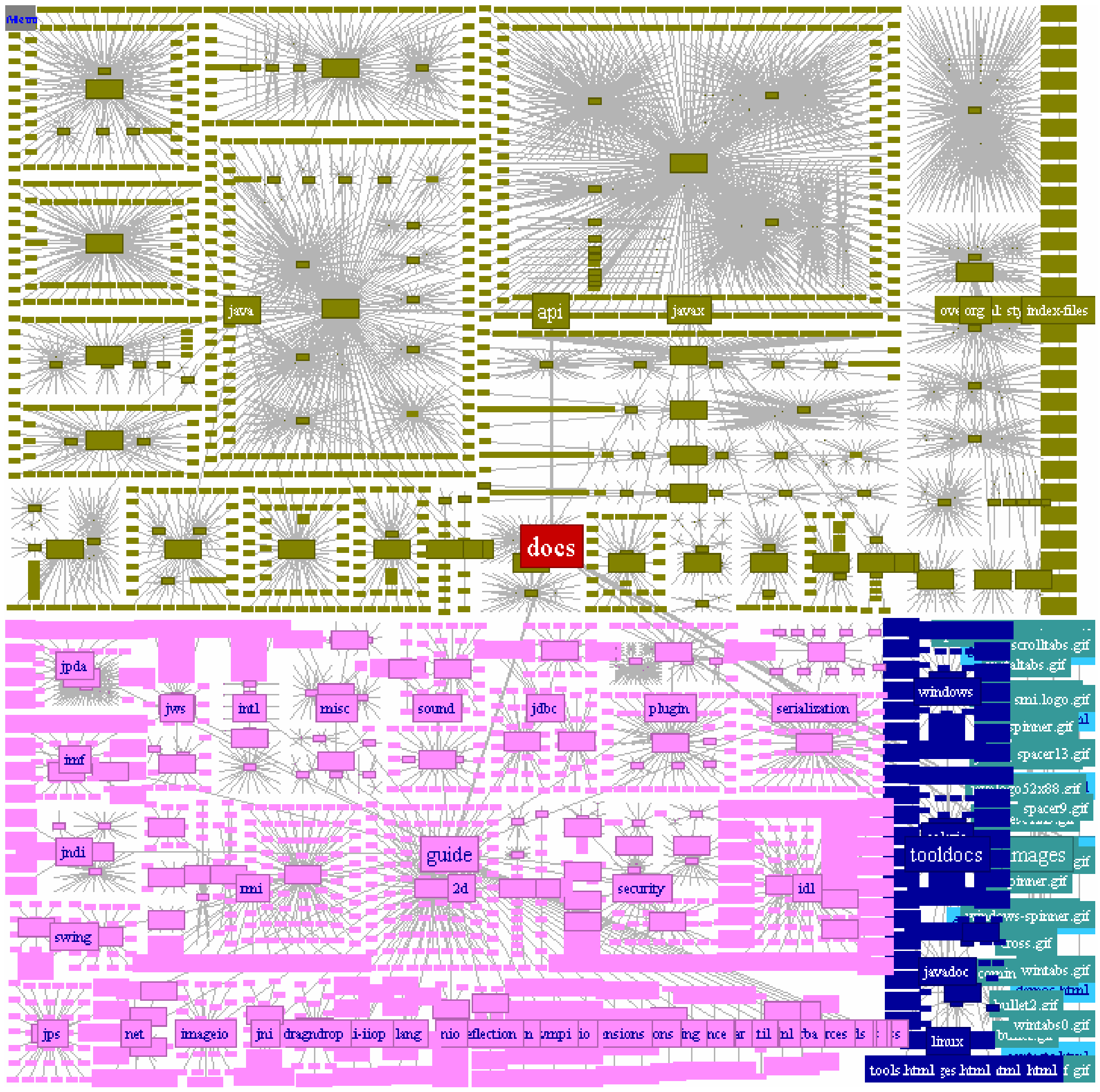


Figure 2.20. An example of the entire Java v.1.4.1 documentation with approximately 9,500 nodes.

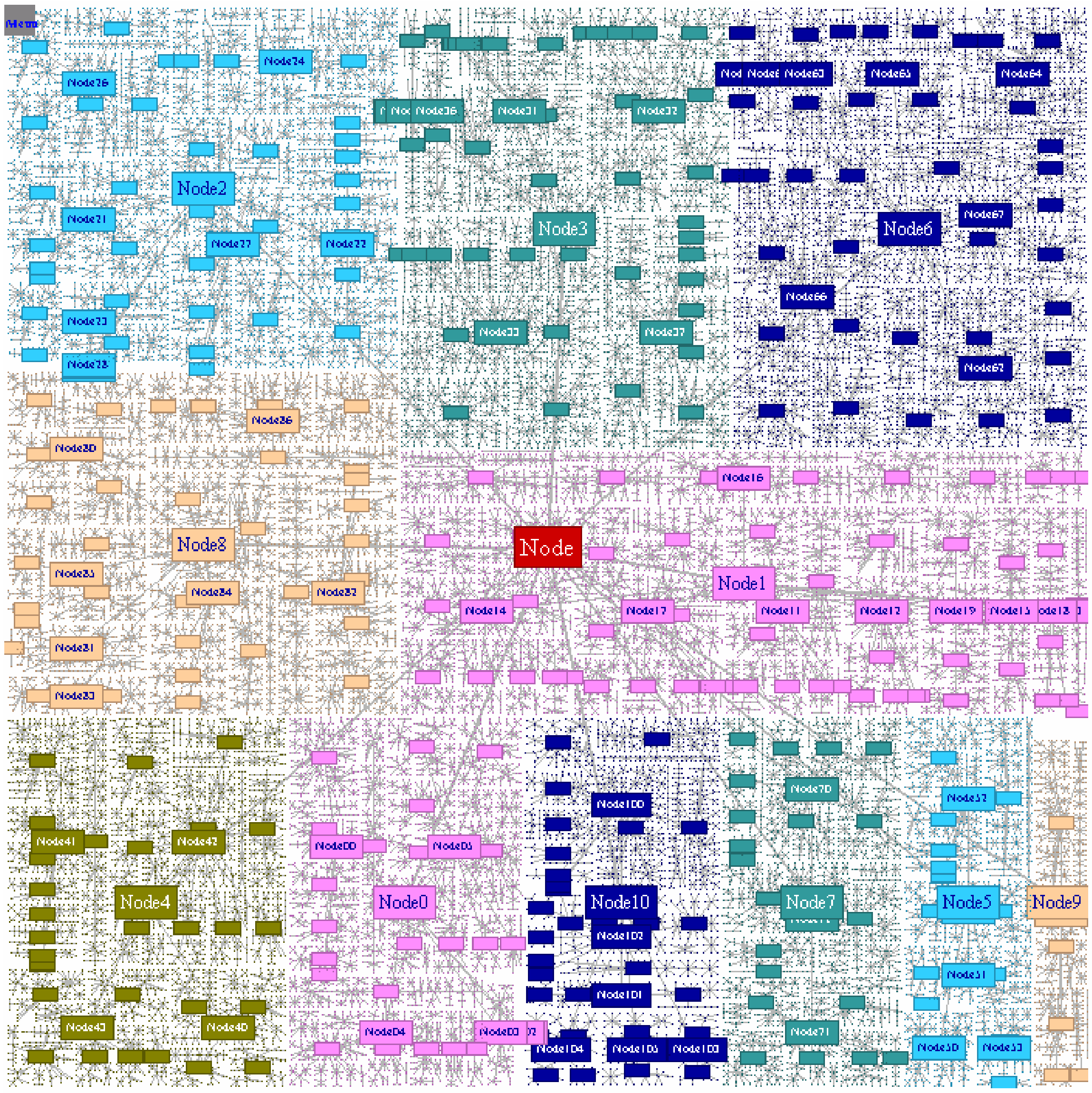


Figure 2.21. An example of a huge dataset of approximately 50,000 nodes.

2.4.6 The Comparison of Different Partitioning Algorithms

The one of the objectives of this research is finding algorithms which can optimise the display space while the layout still satisfies the aesthetical pleasant. Therefore, the partitioning algorithms must address a specific area partitioning criteria, to derive a complete set of rectangular local regions that can produce a high quality layout of the node-link diagram, in terms of satisfying the following general aesthetics rules defined by Di Battista, Eade, Tamassia, and Tollis in their *Graph Drawing* book (Di Battista et al. 1999). Typical aesthetics rules are listed as:

- **Edge Crossings -** edge crossings make it difficult to trace paths. So all edge crossings should be removed if the graph is planar, otherwise the total number of edge crossings should be minimised.
- **Angular Resolution -** the angle between the edges of any vertex should be maximised. This aesthetic rule is especially relevant for straight-line drawings.
- **Total Edge Length -** the sum of edge lengths should be minimum, or the average of all edge lengths should be as small as possible.
- **Uniform Edge Length -** the variance of the lengths of the edges should be minimum. This aesthetic rule includes the minimisation of the number of long edges as well as very short edges. The long edges usually cost extra cognitive effort to perceive the parent-child relationships while the short edge sometimes cause overlaps among the graphical nodes (see Figure 2.22).

Obviously, from Figure 2.22a we can see that the use of thin rectangular local regions for placing node-link diagrams will significantly reduce the quality of graph presentation. Therefore, the *squarification* of local regions is essential in the design of the space partitioning algorithms, especially when node-link diagrams are placed to show relationships among vertices.

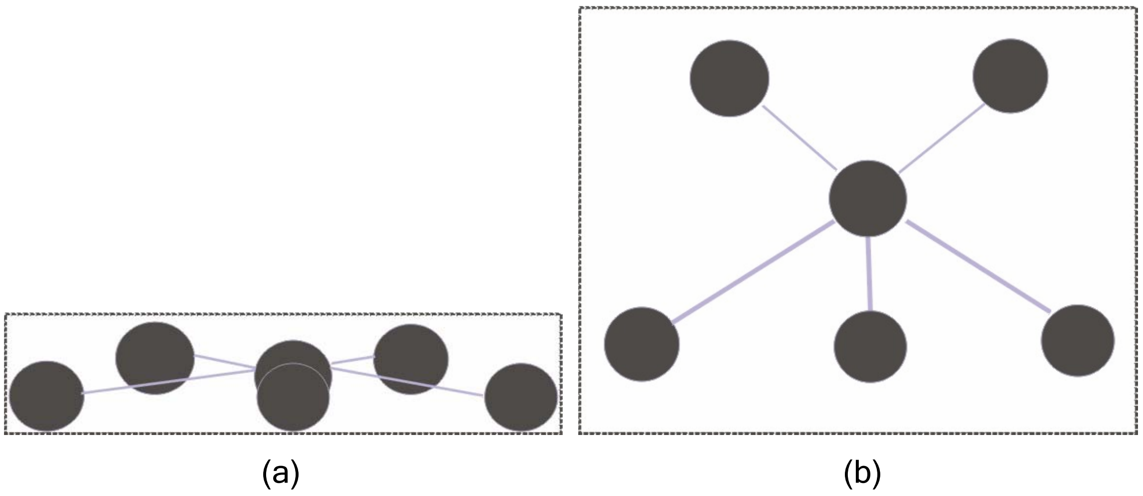
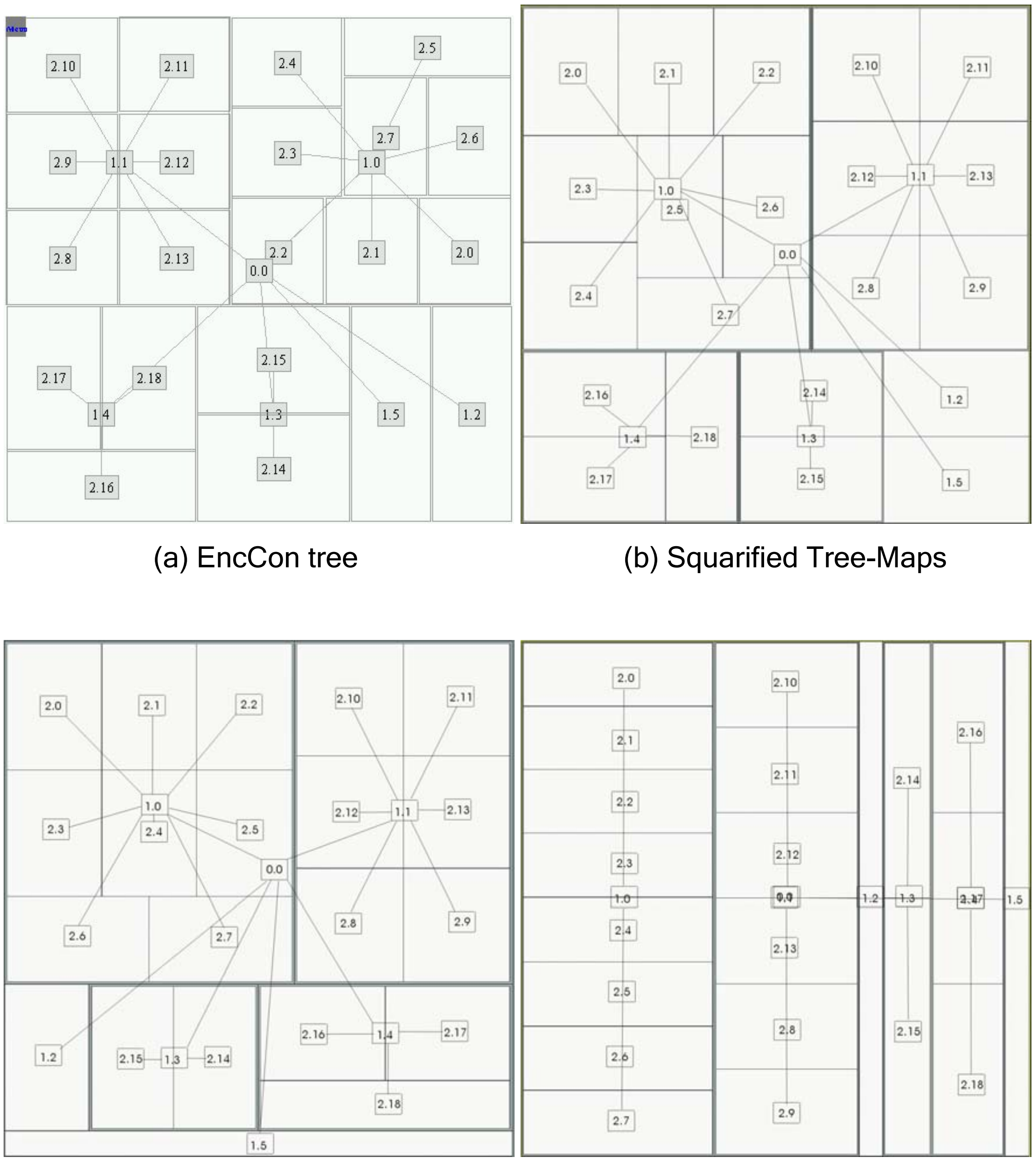


Figure 2.22. An example of node-link diagram at different type of rectangles. The left-hand side image shows an example of placing a node-link diagram in a thin rectangle. We can see that the quality of the graph presentation in this thin rectangle is much lower, in terms of *Angular Resolution* and *Uniform Edge Length* aesthetic rules, than the quality of layout of the same graph presented in a square-like rectangle on the right-hand side. Note that the left-hand side diagram contains an overlapping node.

Figure 2.23 shows an example of space partitioning using four different partitioning algorithms, including *EncCon tree*, *squarified tree-maps* (Bruls, Huizing & van Wijk 2000), *strip tree-maps* (Bederson, Shneiderman & Wattenberg 2002), and *slice and dice tree-maps* (Johnson & Shneiderman 1991). The *EncCon tree*'s vertex positioning rules are then used to place a small node-link diagram in each of these images so that we can observe the quality of these layouts generated by different partitioning algorithms. Clearly, for small datasets the differences between these algorithms, in terms of layout quality, computational time and space utilisation, are not significant. We can see from these images in Figure 2.23 that the output layouts are similar, except for the layout generated by the original tree-maps algorithm as shown in Figure 2.23d. The use of the *slice and dice tree-maps* for the partitioning, the hierarchical node-link diagram is heavily overlapped and hard to be seen. From this figure, we see that the partitioning of the *squarified tree-maps* and the *EncCon tree* produces relatively good layouts for small node-link diagrams. However, the real advantage of the *EncCon tree* visualisation is to present large datasets with high density layouts, where vertices of the graph are well distributed and the space utilisation is maximised (see Figure 2.26 and Figure 2.27). The next section describes an evaluation of these square-like area partitioning algorithms, including the *squarified tree-maps* and the *EncCon tree*, applying the aesthetics rules defined as above. This will enable us to compare these algorithms in terms of producing high quality layouts.



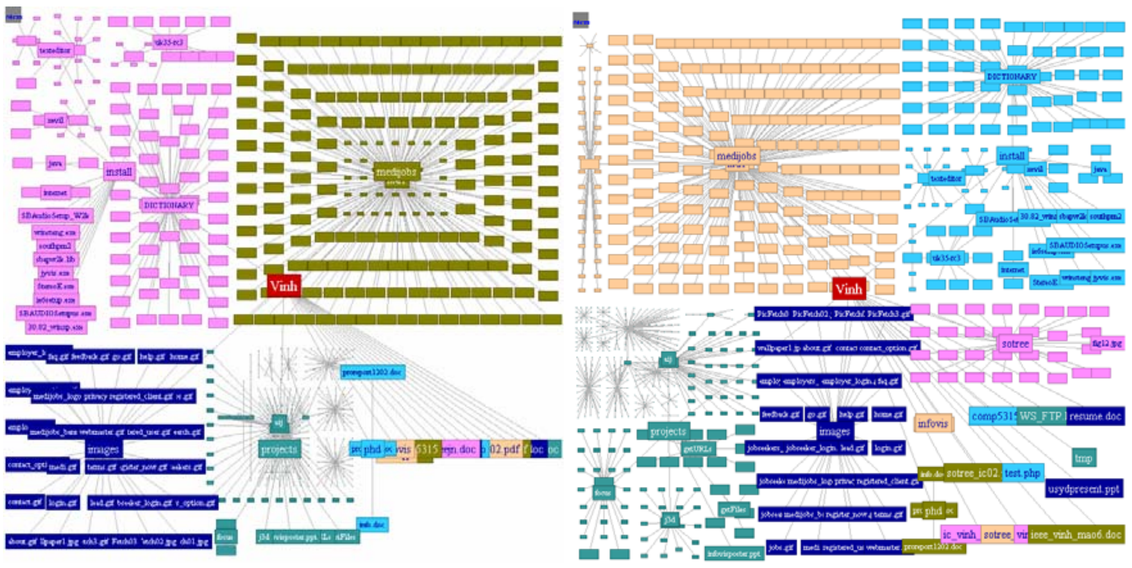
(a) EncCon tree (b) Squarified Tree-Maps

(c) Strip Tree-Maps (d) Original Tree-Maps

Figure 2.23. The layouts of a tree generated by four area partitioning algorithms. Image (a) shows a layout using *EncCon tree'* s partitioning algorithm. Image (b) shows a layout using *squarified tree-maps'* partitioning algorithm. Image (c) shows a layout using *strip tree-maps'* partitioning algorithm. Image (d) shows a layout using the original *slice and dice tree-maps'* partitioning algorithm.

2.4.7 An Experimental Evaluation

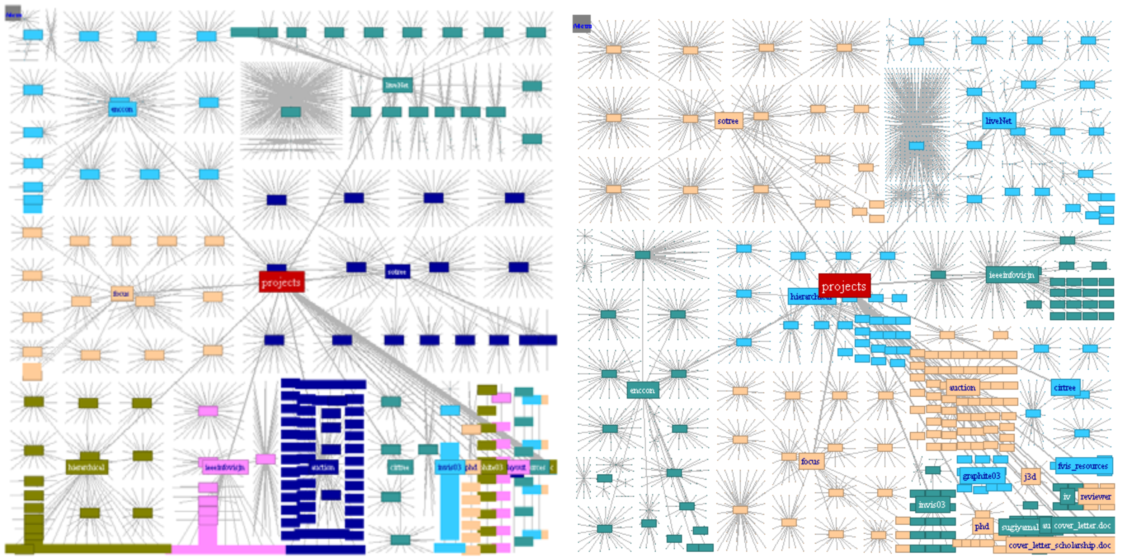
From the comparison of four partitioning algorithms shown in Figure 2.23, we see that the square-like partitioning algorithms, which include *squarified tree-maps* and *EncCon tree*, usually produce better layouts for the placement of small node-link diagrams. Other partitioning algorithms, such as *strip tree-maps* and *slice and dice tree-maps*, are not as good in this kind of visualisation. Therefore, in this section we only evaluate *squarified tree-maps'* and *EncCon tree's* partitioning algorithms, and then compare the outcomes of the evaluation against the aesthetics rules defined at *Graph Drawing* book (Di Battista et al. 1999). This experimental evaluation was carried out with five datasets consisting of 26, 900, 3660, 9500 and 50,000 nodes. The layouts of these datasets in both algorithms are presented respectively from Figure 2.23 to Figure 2.27. The aesthetic rules we are using for the evaluation include *edge crossings*, *angular resolution*, *total edge length*, and *uniform edge length*.



(a) (b)

Figure 2.24. The layouts of the 2nd experimental dataset.

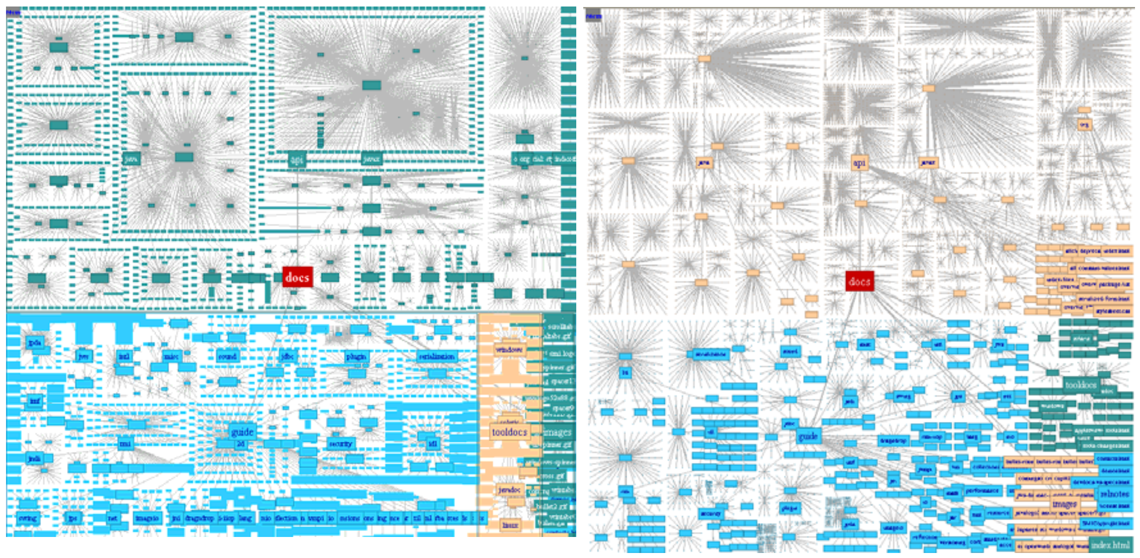
This dataset is an Unix file system with approximately 900 directories and files that are visualised by (a) *EncCon tree's* partitioning algorithm and (b) *squarified tree-maps'* partitioning algorithm .



(a) (b)

Figure 2.25. The layouts of the 3rd experimental dataset.

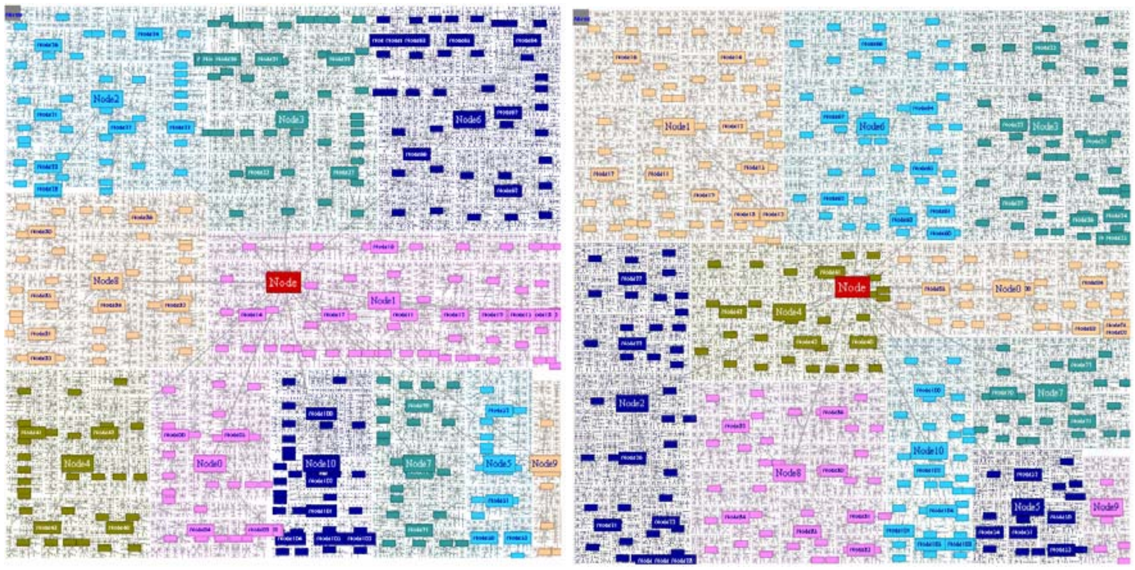
This dataset is a file system with approximately 3,600 directories and files that are visualised by (a). *EncCon tree's* partitioning algorithm and (b) *squarified tree-maps*' partitioning algorithm.



(a) (b)

Figure 2.26. The layouts of the 4th experimental dataset.

This dataset is the entire Java SDK v.1.4.1 Documentation with approximately 9,500 directories and files that are visualised by (a). *EncCon tree's* partitioning algorithm and (b) *squarified tree-maps*' partitioning algorithm.



(a) (b)

Figure 2.27. The layouts of the 5th experimental dataset.

This dataset is a very large data with approximately 50,000 nodes that are generated by (a). *EncCon tree's* partitioning algorithm and (b) *squarified tree-maps*' partitioning algorithm.

In the following subsection, we evaluate the partitioning algorithms against each of these aesthetic rules.

2.4.7.1 Edge Crossings

Edge crossings make it difficult to trace paths and to interpret the relationships. Therefore, all edge crossings should be removed or the number of edge crossings should be minimised. To reduce the complexity of evaluation, we only consider the edge crossings in the top three levels of the hierarchy. This is because those small graphical edges in the lower levels of the hierarchy can only show the global "density" information of the datasets, rather than particular detailed relationships. They are too small to be seen, and crossings among these small edges are not significant, in terms of interpreting particular relationships. Therefore, we can ignore counting these edge crossings in the lower levels of the hierarchy. The following table shows the actual number of edge crossings occurred in five different layouts generated by *EncCon tree* and *squarified tree-maps* algorithms.

	Squarified tree-maps	*EncCon tree*
No. of crossings in dataset 1	0	0
No. of crossings in dataset 2	32	8
No. of crossings in dataset 3	812	276
No. of crossings in dataset 4	44	58
No. of crossings in dataset 5	8	12

Table 1. Edges Crossings of the Top Three Levels.

We can see from the above table that *EncCon tree*'s partitioning algorithm produces fewer edge crossings for datasets 2 and 3, while *squarified tree-maps*' partitioning algorithm produces slightly fewer crossings for datasets 4 and 5.

2.4.7.2 Angular Resolution

The *angular resolution* aesthetic rule measures the *average angular variance* of all angles formed by edges of a non-leaf node and its child vertices. Suppose that vertex v_i has k children $\{v_{l+1}, v_{l+2}, ..., v_{l+k}\}$ and the angle formed by edges $\{v_iv_{l+1}, v_iv_{l+2}, ..., v_iv_{l+k}\}$ are respectively $\{\theta_{l+1}, \theta_{l+2}, ...,\theta_{l+k}\}$. The *average angle* θ_i of vertex v_i is calculated by the following formula:

$$\alpha_i = \frac{1}{k}\sum_{j=1}^{k}\theta_{l+j}$$

Equation 2.10

Then, the *average angular variance* AV_i (in percentage scale) of all the angles $\{\theta_{l+1}, \theta_{l+2}, ...,\theta_{l+k,}\}$ at vertex v_i is calculated by:

$$AV_i = \frac{100\%\sum_{j=1}^{k}|\theta_{l+j} - \alpha_i|}{k\alpha_i}$$

Equation 2.11

Ideally, if all angles at vertex v_i are equal then $AV_i = 0$. Finally the overall *average angular variance* AV of the entire tree T, which consists of n vertices $\{v_1, v_2, \ldots, v_n\}$ is calculated by the following formula:

$$AV = \frac{1}{n}\sum_{i=1}^{n} AV_i$$

Equation 2.12

We aim to minimise the value of AV for the layout of given trees. The following table shows the *angular resolution* in five different layouts generated by *EncCon*'s and *squarified tree-maps'* algorithms.

	Squarified tree-maps	*EncCon tree*
AV value in dataset 1	11.08	15
AV value in dataset 2	42.97	32.42
AV value in dataset 3	43.44	43.35
AV value in dataset 4	48.73	45.28
AV value in dataset 5	30.71	25.88

Table 2. Angular resolution.

As can be seen from the above table, the *EncCon tree*'s partitioning algorithm produces a better *angular resolution* with datasets of 2, 3, 4 and 5, while the *squarified tree-maps*' partitioning algorithm performs slightly better with the small dataset 1.

2.4.7.3 Total Edge Length

To satisfy the *total edge length* aesthetic of graph drawing, the sum of the lengths of all edges of the graph should be minimised. This aesthetic rule is usually measured through the value of *average edge length*. In other word, the average of all edge lengths in optimised graph layouts should be as small as possible.

Suppose that $G = \{V, E\}$ is a graph, where the edge set $E = \{e_1, e_2, ..., e_n\}$. The length l_{e_i} of each edge e_i linking between two vertices $v_a(x_a, y_a)$ and $v_b(x_b, y_b)$ can be easily calculated by the following formula:

$$l_{e_i} = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

Equation 2.13

Then the *average edge length AL* of all edges in graph *G* can be calculated by the following formula:

$$AL = \frac{1}{n}\sum_{j=1}^{n} l_{e_j}$$

Equation 2.14

This aesthetic rule, in conventional graph drawing, is defined based on the assumption that all edges and vertices of a given graph are assigned the same value for their graphical attributes. For example, under this assumption all vertices will use the same style of icons (nodes) with the same size and same background for their visual presentation, and all edges will use the same graphical line-type with the same thickness and the same brightness for their visual presentation.

However, in *enclosure+connection* visualisation, the above assumption no longer holds. In *EncCon tree*, vertices and edges at different levels of the hierarchy are associated with different graphical attributes or different values of the same attributes. Fortunately, all vertices and edges in the same level of the hierarchy are assigned the same graphical attributes and the same value of these graphical attributes.

We believe that it is unfair to count all edges of different levels with different visual effects for the calculation of this single measurement. Therefore, this aesthetic is modified by calculating the *average edge length* AL_0, AL_1, and AL_2 of all edges at different levels, respectively levels 0, 1, and 2, of the hierarchy separately. The following table shows the *average edge length* of five different layouts at the top three levels of the hierarchy, generated by *squarified tree-maps'* and *EncCon tree*'s partitioning algorithms.

	Level 0 – AL_0		Level 1 – AL_1		Level 2 – AL_2	
	Squarified	*EncCon*	*Squarified*	*EncCon*	*Squarified*	*EncCon*
Dataset 1	265.06	263.9	117.25	116.2	0	0
Dataset 2	309.11	300.06	97.6	99.52	105.83	112.08
Dataset 3	339.91	339.86	76.09	87.82	29.59	31.26
Dataset 4	326.91	307.95	155.97	168.44	38.52	48.41
Dataset 5	268.73	279	92.3	92.55	41.29	45.47

Table 3. Average edge length (the size of display is: 700x700 pixies).

2.4.7.4 Uniform Edge Length

Like the *angular resolution* aesthetic rule, the *uniform edge length* aesthetic rule is for the measurement of the average length variance of all edges of a given graph (or a sub-graph). We aim to minimise the variance of the lengths of the edges. An optimised graph layout should have all edge lengths as uniform as possible.

Suppose that $G = \{V, E\}$ is a graph, where the edge set $E = \{e_1, e_2, ..., e_n\}$ with the corresponding edge lengths of $\{l_{e1}, l_{e2}, ..., l_{en}\}$. The length l_{ei} of each edge e_i can be easily calculated by Equation 2.13, and the *average edge length AL* of all edges in graph G can be calculated by Equation 2.14. Thus, the average length variance *ALV* (in percentage scale) of all edges in graph G can be calculated by the formula in Equation 2.15.

$$ALV \; \frac{100\% \sum_{j=1}^{n} | l_{e_j} - AL |}{nAL}$$

Equation 2.15

We also calculate three values ALV_0, ALV_1, and ALV_2 for three different levels of the hierarchy separately because of the above reason. Table 4 shows the *average length* *variance* of five different layouts at the top three levels of the hierarchy, generated by *squarified tree-maps*' and *EncCon tree*'s partitioning algorithms.

	Level 0 – ALV_0		Level 1 – ALV_1		Level 2 – ALV_2	
	Squarified	*EncCon*	*Squarified*	*EncCon*	*Squarified*	*EncCon*
Dataset 1	16.68	16.33	29.78	32.14	0	0
Dataset 2	20.21	16.04	42.27	35.63	46.72	38.92
Dataset 3	24.79	18.56	55.13	40.46	37.57	38.05
Dataset 4	31.27	27.64	66.15	51.14	61.61	58.86
Dataset 5	25.7	23.05	38.78	32.7	40.75	42.61

Table 4. Average length variance (the size of display is: 700x700 pixies).

2.4.7.5 Experimental Results and Discussion

The results of this evaluation, based on the above criteria and five datasets, are summarised at Figure 2.28. This figure shows a comparison of the aesthetic rules, including *edge crossings*, *angular resolution*, *total edge length*, and *uniform edge length*, between two layouts *squarified tree-maps* and *EncCon tree*. Five experiments were carried out with the five different datasets, covering small, medium, moderately large, large and very large hierarchical data examples (see those layouts from Figure 2.23 to Figure 2.27).

The result of the above comparisons is that for medium and moderately large datasets, *EncCon tree*'s partitioning performs significantly better than *squarified tree-maps* in terms of minimising edge crossings. The layout of a medium (or moderately large) size graph generated by the *EncCon tree*'s partitioning algorithm contains only approximately 25% of the edge crossings which occur in the layout of the same graph generated by the *squarified tree-maps*.

We can also see that *EncCon tree*'s partitioning produces a better *angular resolution* in layout of datasets 2, 3, 4 and 5, while *squarified tree-maps*' partitioning produces a slightly better *angular resolution* in layout of dataset 1 with a small number of nodes.

There is no significant difference between these two algorithms in the aesthetical criteria *average edge length*. However, *EncCon tree*'s partitioning does significantly reduce the *average length variance* in the layouts of moderately large or large graphs.

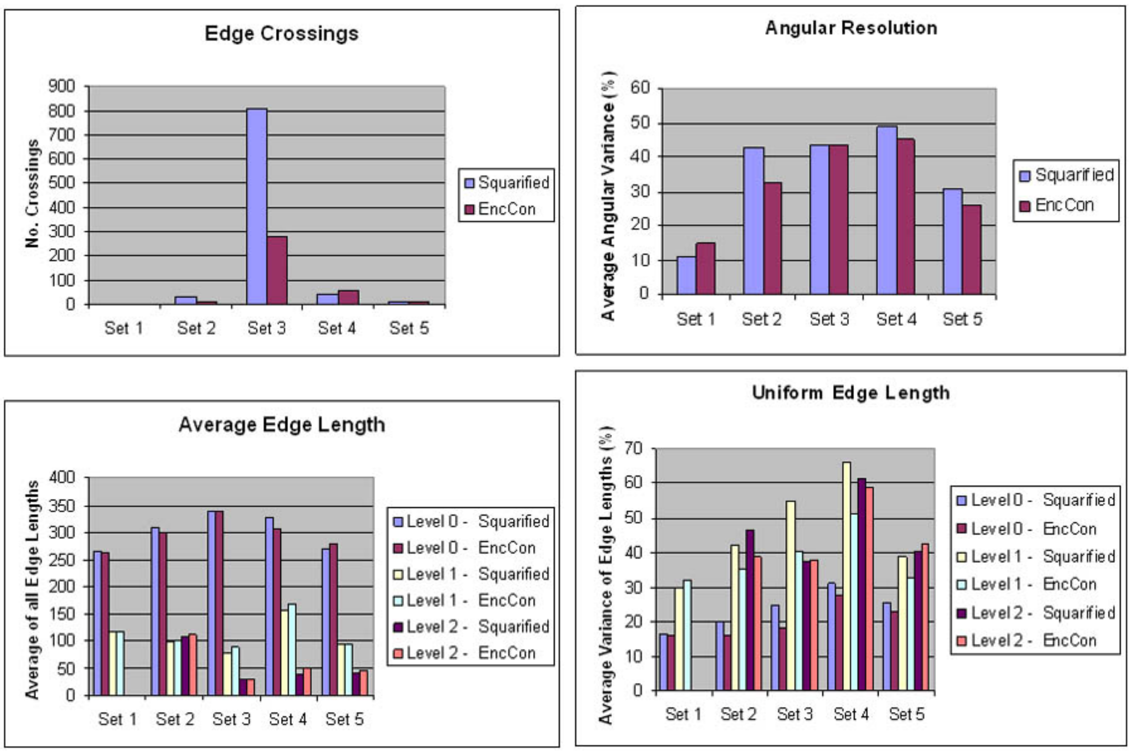


Figure 2.28. A summary of the experimental evaluation results.

2.5 Summary

This chapter has presented a new visualisation model, called *enclosure+connection* that is different from the traditional methods. It can not only ensure the efficient utilisation of display space, but also show the relational structure explicitly by using a node-link diagram.

We have introduced two new geometrical layout algorithms, *space-optimised tree* and *EncCon tree*, to support our new model. Both algorithms are efficient for visualising large hierarchies. Technically, the *space-optimised tree* layout algorithm uses a polygonal partitioning for drawing trees, while the *EncCon tree* uses a simple rectangular partitioning. The rectangular partitioning method used in the *EncCon tree* is similar to the method used in *squarified tree-maps* and both methods try to produce a sequence of square-like rectangles. Similarly to the *space-optimised tree*, the *EncCon* *tree* can also maximises the utilisation of display space. However, its rectangular partitioning provides users with a more straightforward way to perceive the relational structures.

To illustrate the technical advantages of *EncCon tree's* partitioning algorithm, we conducted an experimental evaluation between the *EncCon tree* and the *squarified tree-maps*' partitioning algorithms based on four aesthetic rules. The outcome of the evaluation shows that the *EncCon tree*'s algorithm usually performs better than the *squarified tree-maps*'s algorithm, especially for partitioning large datasets.

Chapter 3 Navigation Methods

Another major problem of the hierarchical visualisation is the navigation which is defined as how to use visual interactions to find particular data items in the information space. Although some existing navigation techniques, as discussed at section 1.3.2, have been developed to facilitate the retrieval and viewing of data items in medium size information spaces, the exploration of large information spaces remains a challenging task in the design of graph and/or hierarchical visualisation. Even if advances in graph visualisation are able to geometrically place and display large graphs and/or hierarchies on the screen, the retrieval of actual data items through the visualisation would still be impossible unless an effective navigation mechanism is provided. For example, without navigation how could we find a particular data item from an efficient visualisation of very large hierarchy of thousands of nodes? (See examples at Figure 2.8, Figure 2.10 Figure 2.20, and Figure 2.21)

Visual navigation is basically a mechanism that constantly guides users to jump from one part of the visualisation to another, allowing them to move quickly around the corresponding information space to find and access particular pieces of information. An efficient navigation mechanism with a set of clear navigational views will help users to gain an idea of the scope of the information source, to find where they are in the information space, to go where they can find information they want, and to go back to where they have been. When the visualisation is created, the design of associated navigation is a crucial task that should also be carried afterwards. Without providing efficient navigation mechanism in collaborating with the large display, the visualisation technique is useless in the field of information retrieval that aims to assist users to retrieve particular data items they want.

Focus+context viewing is one of the most advanced and commonly used navigation technique cooperated with many existing visualisation systems. This thesis also focuses on the use of the principle of this approach and the investigation of alternative *focus+context* methods.

Chapter 2 has described the space-efficient visualisation model and two associated layout algorithms which for visualising large hierarchies. These algorithms can effectively display the entire of a hierarchy within a limited display screen. They utilise the space to allow the display of a large amount of information on one screen (see an example at Figure 2.21). Although the original visualisations produced by these layout algorithms can provide an overall view of the large hierarchy, these visualisations alone do not provide the detailed view of a particular subset and they do not help at all for finding particular data items the user wants. Thus, the navigation mechanisms are necessary to enable users browsing through the large hierarchy.

In this chapter, we concern with the problem of how to navigate through such large visualisation. We describe the technical detail of two main navigation algorithms, called *hybrid view,* and *layering view*. These methods are both *focus+context* approaches and are independent to the layout algorithms although they are implemented based on the prototypes of respectively *space-optimised tree* and *EncCon tree*. Thus, all the figures and examples from the *hybrid view* and the *layering view* are built respectively from the prototype of the *space-optimised tree* and the *EncCon tree* layout algorithms. This is due to the historical reason when these interactive navigation methods were developed.

The *hybrid view* technique includes a *browsing algorithm* and a *distortion algorithm*. The *browsing* technique is responsible for bringing a sub-hierarchy to the focus region while the *distortion* technique is used to enlarge information at the focus area. This technique also employs a *semantic zooming* to enhance the navigation when the visualisation is too overcrowded. On the other hand, the *layering view* technique uses the *semantic zooming* and *layering* to achieve the *focus+context* visualisation. Technically, it employs semi-transparency to archive the display of layers at the same display space. This allows both context view and detailed view to be displayed at two separate layers in an overlapping manner at the same physical screen space. The detail of these two interactive navigation methods is described below.

3.1 The Hybrid View

Hybrid view includes a *focus+context* technique called *DualView* and a *semantic zooming* technique. The *semantic zooming* is applied to the situation when a very high dense visualisation is displayed in which the ordinary distortion are hard to be implemented because of a large number of nodes and edges. Particularly, this is because that the computational complexity is significantly high for calculating a large number of distorted nodes and edges. Furthermore, the high density of nodes and edges at the focus view might also make it difficult to perceive the information. The *DualView* provides an interactive *focus+context* solution to help users browse through the hierarchy when the visualisation is not very dense.

Sematic zooming can be used to enlarge a focused sub-tree and filter out the others. This interactive zooming responds to the mouse-click event. When a node is clicked, this focused node moves toward to the centre of the display area. The layout of its sub-tree then expands to the entire display space accordingly. Technically, the layout of sub-tree is recalculated for its new geometrical position. All other sub-trees are disappeared in the new visualisation. However, the direct-ancestors of the selected node are still displayed at the history path for keeping track of the current navigation. This path displays all direct-ancestors of the selected node in order. We also assign different graphical properties to these ancestors to distinguish them from the normal nodes. The user can easily move back to the visualisation of an upper level hierarchy by simply clicking on an ancestor node along the history path. Figure 3.1 shows an example of the *semantic zooming* in *space-optimised tree* when a sub-tree is selected to be viewed in detail.

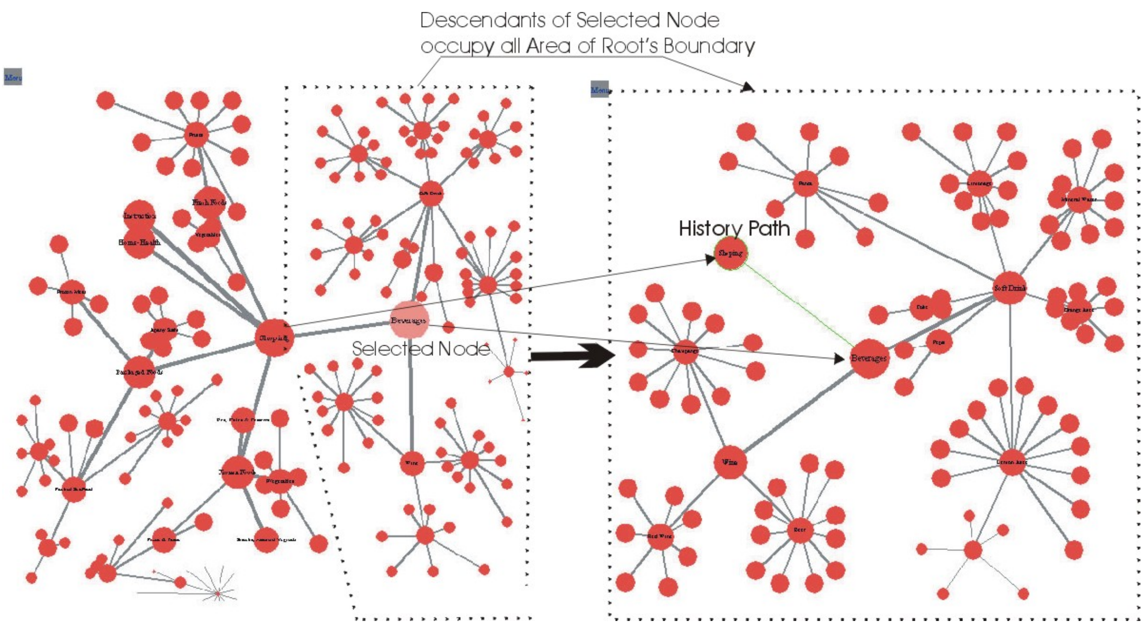

Descendants of Selected Node occupy all Area of Root's Boundary
History Path
Selected Node
Beverages
Shopping
Soft Drinks
Wine
Beer

Figure 3.1. An example of semantic zooming when a sub-tree is selected to be viewed in detail.

Animation is used to preserve a mental map when the view is changing. It is a real challenge to implement animation in viewing very large hierarchies. The expensive computational cost of view transformations of large datasets (with more than 10,000 nodes) might disrupt the smooth animation. Therefore, to reduce the computational cost we only apply animation to those nodes that are graphically visible within the screen resolution. This property ensures that the number of nodes involved in animation is reduced to a few tens or hundreds.

Semantic zooming is an excellent technique for navigating very large hierarchies. However, the lack of displaying contextual information might prevent users from viewing other parts of the hierarchy. In order to overcome this problem, we also use *DualView*, a *focus+context* technique, as a supplementary technique for the navigation of large hierarchies. This method applies a simplified *fisheye* distortion transformation to speed up the calculation of views.

DualView technique technically includes two transformations, the *browsing* and the *distortion*. We use *browsing* transformation to bring interested information into the focus region while a fisheye-like *distortion* transformation is applied to increase the magnification of information at the focus area. These two transformations are applied independently onto both horizontal and vertical directions. The *browsing* and the *distortion* transformations are also independent to the layout algorithms, and thus this technique can be applied to any tree visualisation system. In order to reduce the computation cost, we only apply the transformation functions to visible nodes on a layout and edges are all redrawn as straight lines. Two independent transformations are then applied to every node when the user interacts with our system. Thus, the final transformation of *DualView* is defined by Equation 3.1.

$$T_{DualView} = T_{browsing} \circ T_{distortion}$$

Equation 3.1

There are three types of coordinate systems used in the transformations including *normal coordinate*, *browsing coordinate*, and *distortion coordinate*. The coordinate system of the visualisation in the normal display is called the *normal coordinate*. The coordinate system in *browsing* transformation is called the *browsing coordinate*. And the coordinate in *distortion* transformation is called the *distortion coordinate*. The coordinate of a point in the *normal coordinate*, *browsing coordinate* and *distortion coordinate* are called (x_n, y_n), (x_b, y_b) and (x_d, y_d) respectively.

The following sub-sections explain the technical details of the *browsing* and *distortion* transformations.

3.1.1 Browsing Transformation

Browsing technique is used to transform the visualisation to a new location when the mouse drags a node from one point to another. This transformation is independent of the layout algorithm, and the movement of a point must satisfy the following properties:

- If a point is near to the centre, it will move faster then a point which is far from the centre (see Figure 3.2).
- All points in the displaying area must not move outside the display area during this transformation.

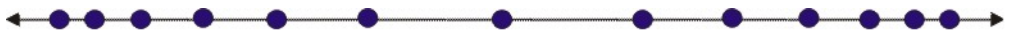


Figure 3.2. An example of the movement of points.

In the *normal coordinate* system, suppose that the mouse drags a point $A(x_{no}, y_{no})$ to a new position $B(x_{no'}, y_{no'})$; and thus a node $N(x_n, y_n)$ in the tree visualisation will move to a new location with coordinate of $(x_{n'}, y_{n'})$. The values of $x_{n'}$ and $y_{n'}$ are calculated independently based on the *browsing* transformation function called $T_{browsing}$.

Suppose that in horizontal direction, the function $T_{browsing}$ is called $T_{browsing}(x_b)$ where x_b is the x-coordinate in the *browsing coordinate* system. The movement property ensures that the value x_b of point *B* in this coordinate system has to satisfy conditions: 1) function $T_{browsing}(x_b)$ is continuous, and 2) $T_{browsing}(x_b)$ towards negative or positive infinites when x_b closes to its minimum or maximum boundaries. Although there are a number of functions which satisfy the above conditions, we simply use the equation:

$$T_{broswing}(x_b) = Tangent(x_b)$$

Equation 3.2

where value of x_b varies in the domain of $(-\frac{\pi}{2}, \frac{\pi}{2})$, and thus $T_{browsing}(x_b)$ varies in the domain of (-∞, ∞).

When x_{n0} moves to $x_{n0'}$ in the *normal coordinate* system, correspondingly x_{b0} moves to $x_{b0'}$ in the *browsing coordinate* system. The function $T_{browsing}(x_b)$ then becomes $T_{browsing}(x_b) + K$, where K is the offset value and is defined as $K = T_{browsing}(x_{b0'}) - T_{browsing}(x_{b0})$. As a result, the new value of x_b is recalculated by the equation:

$$x_b = Tangent^{-1}(T_{browsing}(x_b))$$

Equation 3.3

3.1.2 Distortion Transformation

Distortion magnification is used to enlarge the focus area when the content in this area is too dense. The *browsing* transformation is responsible for moving and enlarging the focus information. However, its magnification sometimes is not sufficient enough to display clearly information in the focus area. Thus, a distortion transformation is applied so that the focus point is at the centre of the display area. Similar to the *browsing* transformation, the *distortion* function $T_{distortion}$ is applied independently in the horizontal and vertical directions. We normalise the *distortion* coordinate to [-1, 1]. In its half domain [0, 1], $T_{distortion}(x_d)$ is a curve that goes through two points (0, 0) and (1, 1) and above the straight line $l(x_d) = x_d$. The transformation function $T_{distortion}(x_d)$ has the same property at other half domain of [-1, 0] that is a curve that goes through two points (0, 0) and (-1, -1) and below the straight line $l(x_d) = x_d$. Simply, our transformation is the function of an arc that goes through 3 points (0, 0), (1, 1) and $C(x_{d0}, y_{d0})$, where $C(x_{d0}, y_{d0})$ defines the distorted magnitude of its magnification (see Figure 3.3).

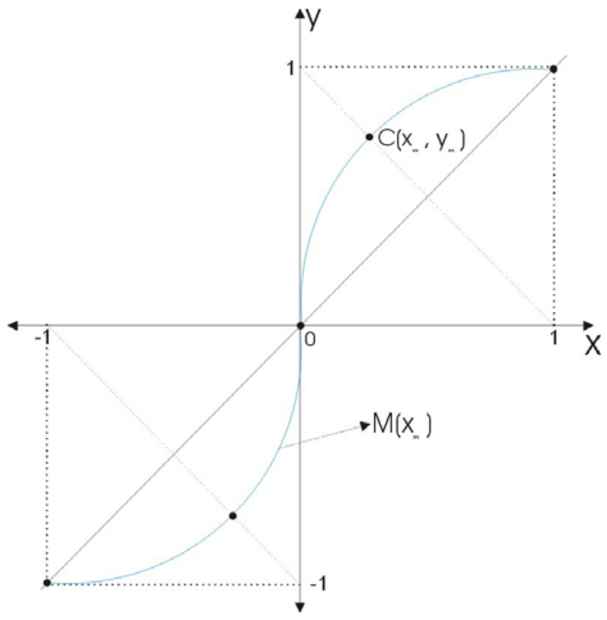


Figure 3.3. Graph of the distortion transformation.

In our prototype, the magnification is simply defined as:

$$T_{distortion} = b + \sqrt{c - (x_d - a)^2} \quad for \quad x_d \geq 0$$

$$T_{distortion} = -b - \sqrt{c - (x_d + a)^2} \quad for \quad x_d < 0$$

Equation 3.4

where a, b, and c are constants.

3.1.3 The Hybrid View's Examples

Images from Figure 3.4 to Figure 3.7 illustrate the examples of the *Hybrid View* navigation technique applied on very large trees. Figure 3.4 shows the entire tree layout which is too dense for using the *DualView*, a *focus+context* navigation technique. Figure 3.5 shows the sub-tree layout when we apply the semantic zooming to enlarge the focus sub-tree and reduce the amount of displayed information. Figure 3.6 shows the layout when we apply the *browsing* transformation to bring interested information into the focus region. Figure 3.7 shows the layout when we apply the *distortion* transformation to increase the magnification to the information at the focus region (i.e. around the central region). This transformation is applied when the information at the focus area is still dense.

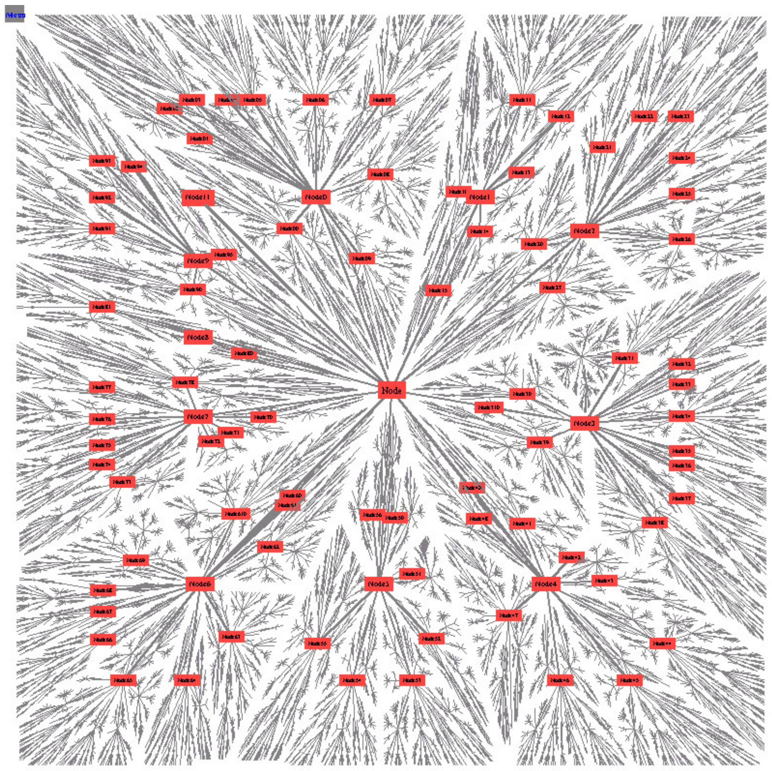


Figure 3.4. The visualisation of an entire tree layout.

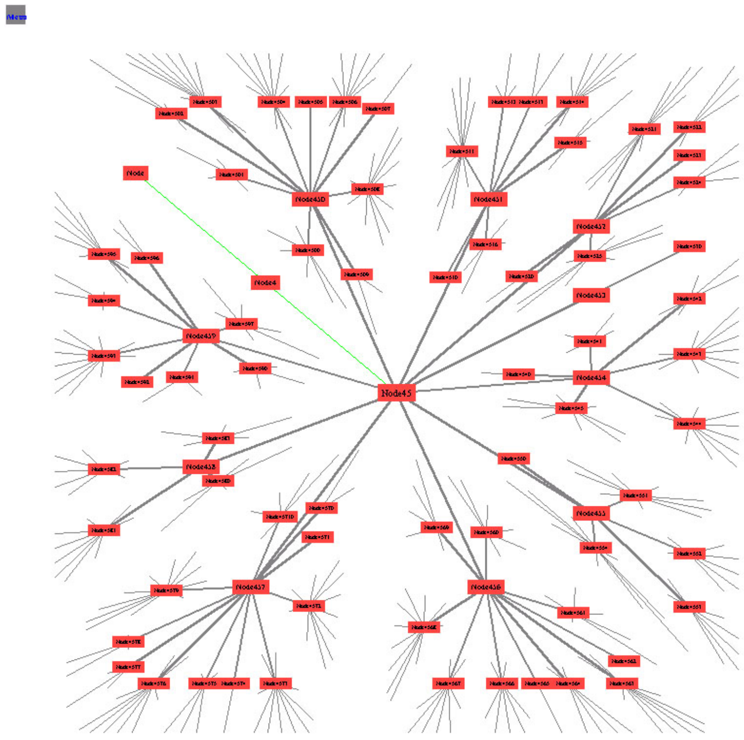


Figure 3.5. The visualisation of a sub-tree of the entire tree shown in Figure 3.4 when semantic zooming is applied.

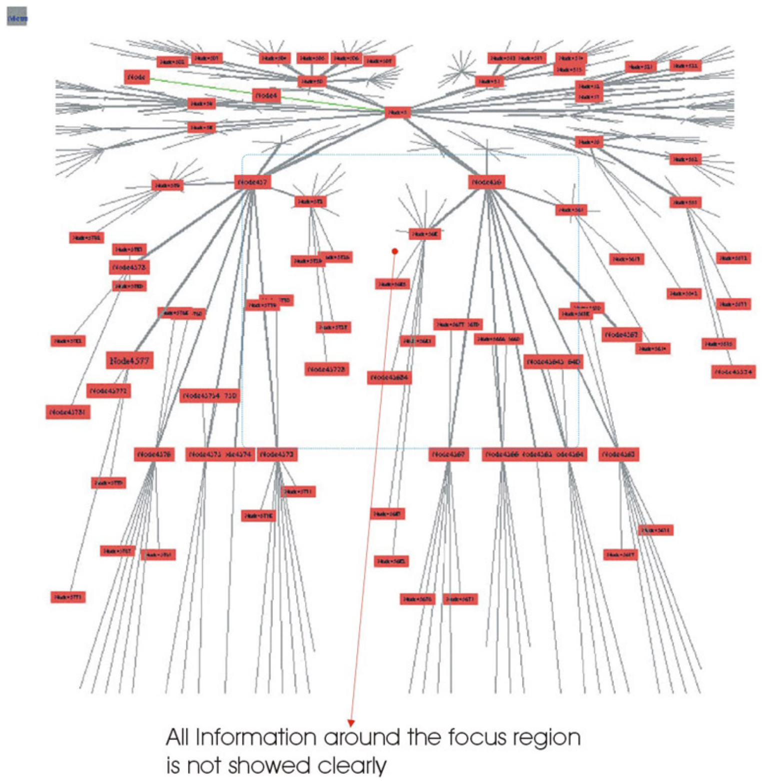


Figure 3.6. The same visualisation as shown in Figure 3.5 when a *browsing* transformation is applied.

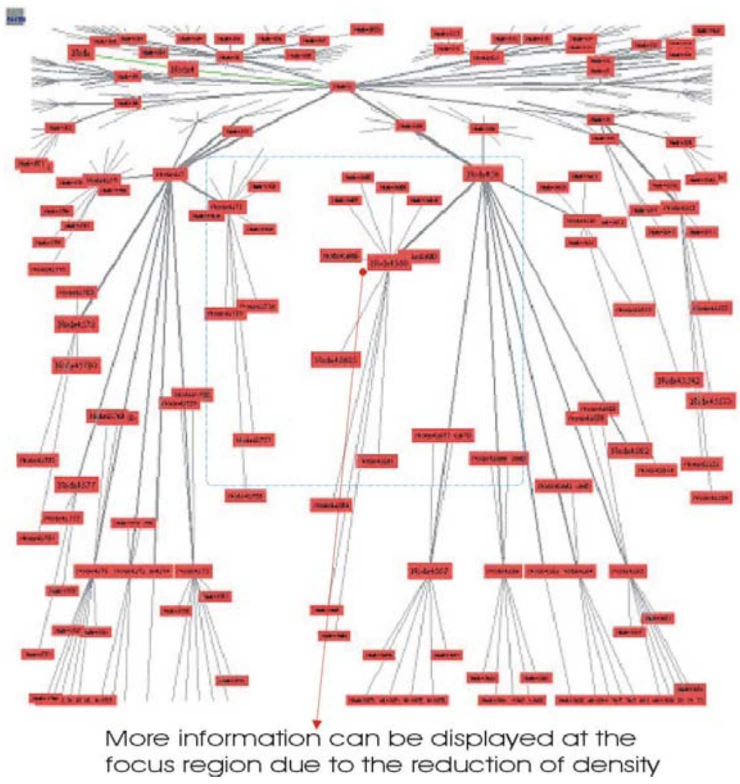

More information can be displayed at the
focus region due to the reduction of density

Figure 3.7. The same visualisation as shown in Figure 3.6 when a *distortion* transformation is applied.

3.2 The Layering View

Most existing *focus+context* techniques uses an *enlarge+embedding* approach which requires the division of the display area for the display of the *global view* (or *context view*) and the *detailed view* (or *focus view*). Examples of these techniques include the *information slices* (Andrews & Heidegger 1998) and *bifocal tree* (Cava, Luzzardi & Freitas 2002). Thus, the display space available for displaying the global structure as well as the focus view is limited. Similarly, other techniques might also display the *context view* and *focus view* of information using a number of separated display windows, which are typically called *multiple-views* technique (Baldonado, Woodruff & Kuchinsky 2000; Convertino et al. 2003; C North 2001; Robert 1998). However, the use of separate windows might break the context (or nature connection) between two views. This costs viewers extra cognitive effort to link two views into one mental map. Therefore, researching a new method to overcome the limitations of the above techniques is necessary for the visualisation of large hierarchies.

This section describes a new *zooming+layering* approach which can achieve the *focus+context* navigation with enlarged *focus view* and *context view*. This technique is different from the traditional *enlarge+embedded* approach (see Figure 1.23). Technically, it employs a semi-transparent graphical technique to achieve the visualisation of two layers of information in the virtual z-coordination of the display space. This allows both *context view* and *detailed view* to be drawn at two separate layers in an overlapped manner on the same physical screen space. This technique always keeps one view highlighted and another not highlighted. The technical details of *zooming+layering* viewing technique are described in the following sections, including layering display and interactive navigation.

3.2.1 Layering Display

Layering display applies semi-transparency to display both *global view* and *detailed view* at two separate layers in the same display space (see Figure 3.8). The *detailed view* is displayed in a full display screen while the *global view* is displayed in a small screen at the background. The system allows the users interactively to shift their focus by switching the view to the front or back between the two views. This technique aims to improve the utilisation of the display space while still providing both the focus and the context information. The *layering display* technique is independent of the layout algorithm, and thus, it can be applied to any layout algorithm.

In our visualisation, the *global view* is overlapped with the *detailed view* and these views are displayed in two separate graphical layers of the same physical screen with different visibility values. As the default mode of the display, the *focus view* is drawn in full screen size on *layer 1* (L_1) of the display while the context view is drawn in a smaller size at the centre on *layer 2* (L_2) (see Figure 3.8). The display on L_1 is defaulted with normal colours and L_2 is defaulted with semi-transparent colours. The purpose of this arrangement is to reduce the distraction between two views. The distraction might occur when too many overlaps of nodes and edges between the two views. The size of the global view can be interactively adjusted to suit the user's preference. In our application, the default size of the global view is half of the entire display area. An example of layering display using transparency can be found at Figure 3.9. Although the *layering display* is independent to the layout algorithms, it works best for connection based visualisation techniques, which use a node-link diagram to present the relationships. This is because that the connection based techniques often leave a large portion of unused space. Thus, the overlaps among nodes between the *context view* and *detailed view* is not much significant.

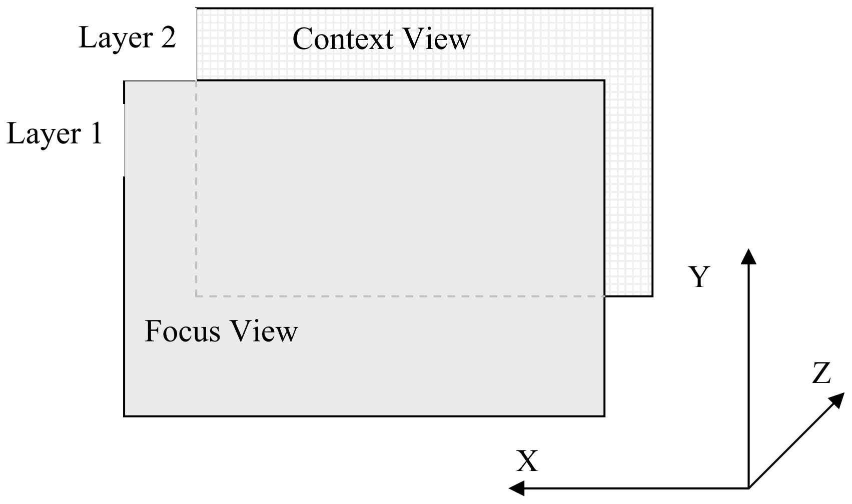


Figure 3.8. Layering display.

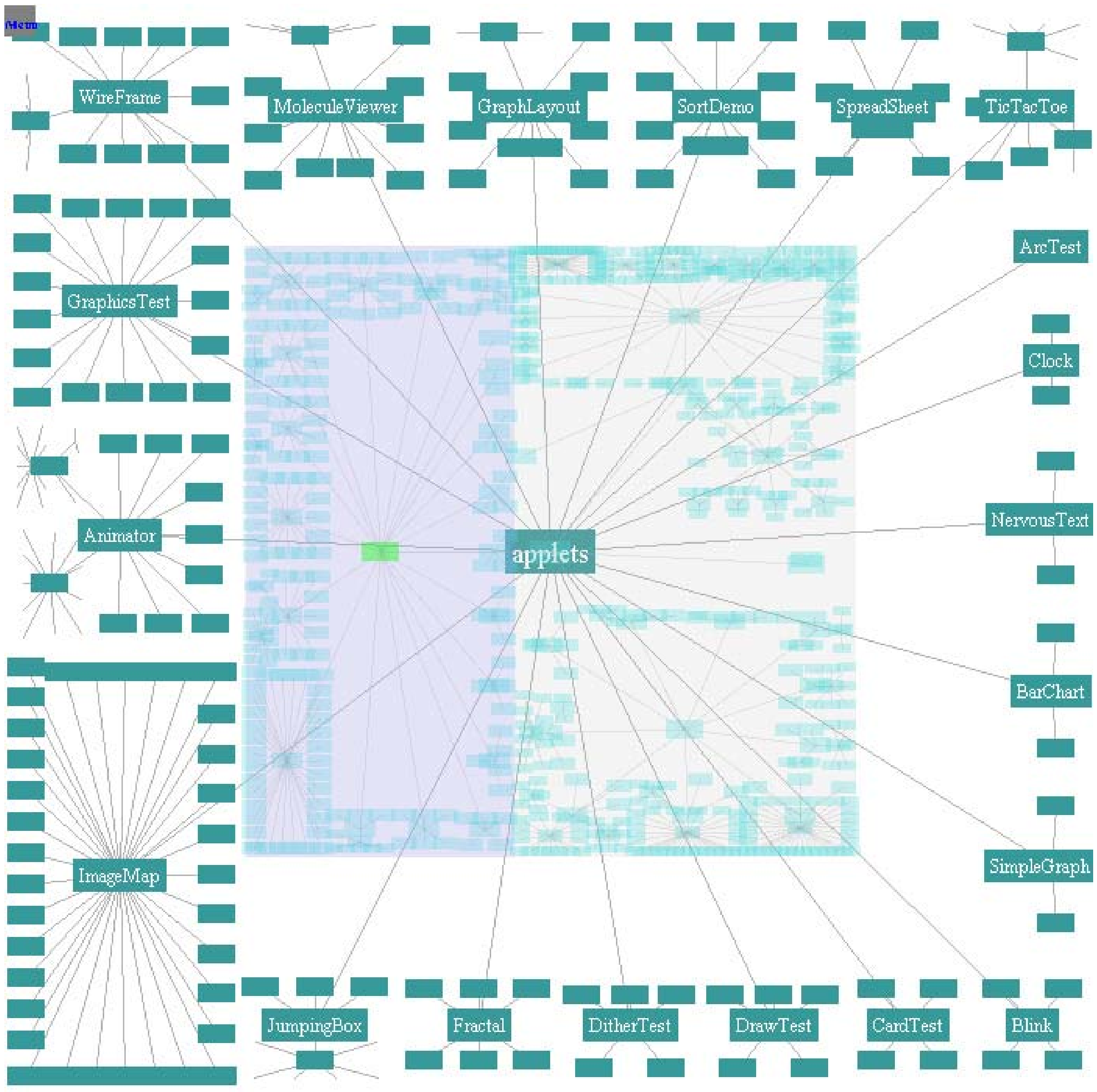


Figure 3.9. An example of focus+context viewing of a file system using semi-transparency technique.

Both of the views are displayed at the same area, the context view is deemphasised and displayed transparently within the small area at the centre and the detail view is focused and occupies the entire display area.

The navigation in the *layering display* is achieved interactively by semantic zooming, updating views and swapping views between two layers. All these transactions are accommodated by animation to preserve the users' mental map of views.

There are two modes of the display: 'default' and 'context'. In the 'default' mode, we assume that the users' attention is on the content of a particular detailed sub-structure from the entire information structure. However, it is quite possible that the users move their attention from a detailed sub-structure to the content of the global structure during the navigation. Therefore, we defined a ‘context’ display mode that shifts the visibility values between two display layers. Practically, we reassign a visible value to L_2 (i.e. the *global view*) allowing users to see the content of the global structure and change the current visible layer L_1 (i.e. the *detail view*) to semi-transparency. In the ‘context’ mode, the display is reversed so that the *global view* is brought from the back to the front and highlighted while the *detail view* is sent from the front to the back and displayed in a semi-transparent manner (see Figure 3.10d). These two views can be shifted interactively by using a left mouse-click on the background of each layer. The shift between views is accommodated by fade in/out animation to preserve the users' mental map of views.

The background of the *context view* is painted with slightly darkened colour in comparison with the detail view's background. This helps the *context view* to stand out from the *focus view*. The selected sub-hierarchy is also highlighted in the *context view* by using a different background colour as well as a selected node (see Figure 3.10b and Figure 3.10d). This property helps to improve the clarity of the display. The *context view* in this visualisation can be either the view of the entire hierarchy or the view of a sub-hierarchy which is previously displayed as a detail view before a user selects a particular subset from this context. Showing the entire context makes it easier to navigate through the hierarchy. However, the major problem arises when this technique is applied for a very deep hierarchy. In this case, a focus sub-hierarchy might become small or even unidentified from the whole context if it locates at several levels away from the root node. On the other hand, showing only the most current history information can overcome the above problem, but it will loose the whole context. Therefore, the choice of which approach to use is dependent on the nature of each application, in order to archive a better result in the navigation. The second method is applied in our demonstration prototype.

3.2.2 Interactive Navigation

In our visualisation, the selection of a focused visual node is the main mode of interactions taken by users during the navigation. This interaction can be applied to any visible vertex v at the *detail view* or *context view* in order to utilise the semantic zooming technique to enlarge the display of sub-tree $T(v)$ into the full screen and bring it to L_1 as a new *detail view*, which overrides the previous display of this layer. The system can also return the *context view* back to the *detail view* by a right mouse click. The *context view* then will be enlarged through semantic zooming and can be brought from L_2 to L_1 for full screen display. The types of interactive navigation and their associated animation are described below.

3.2.2.1 Navigation in the Detail View

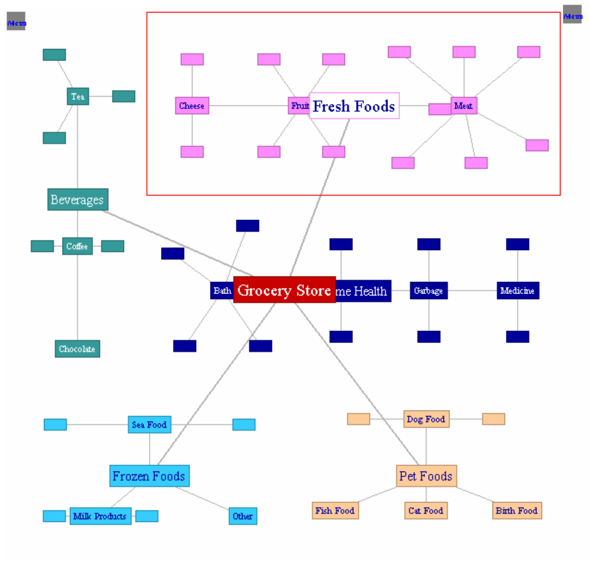
When a vertex v of a sub-tree T in L_1 is selected, its sub-tree $T(v)$ expands smoothly to occupy the entire display area by using *semantic zooming*. Accordingly, the size of the node of its sub-tree increases to match the consistency of the new view. The previous layout of the hierarchy T is reduced and overwritten to L_2 simultaneously, and T now replaces the old context view in L_2 smoothly by using fade animation. The colour of the nodes in the previous display of T will now become semi-transparent and lighter. Figure 3.10c shows the animation of the intermediate state of the transition from Figure 3.10a to Figure 3.10b.

3.2.2.2 Navigation in the Context View

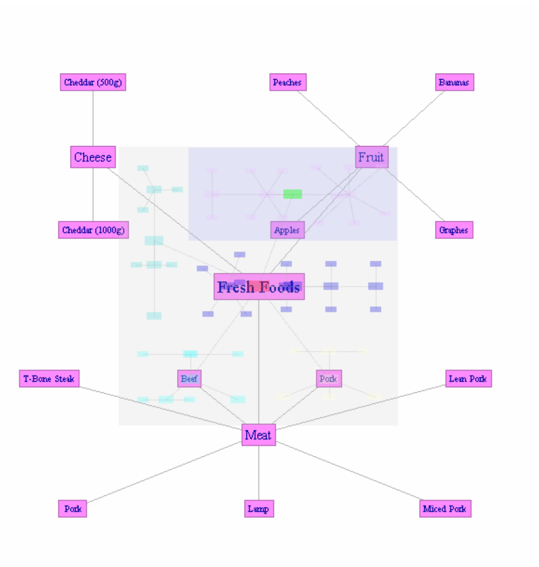
When a mouse click occurs on the local region of the root of the context view of sub-tree T in L_2, the focus is switched to T in L_2. The layout of the context view T is now activated and highlighted (see Figure 3.10d). The user then can select any node in T to enlarge a particular sub-hierarchy of T into the full screen display by using *semantic zooming*. When a node at the context hierarchy T is selected, its entire sub-hierarchy expands from the context region to the detail region occupying the entire display area. In addition, the indication of selected sub-hierarchy at the history layer L_2 is also changed to selected node (see those images at Figure 3.10: (d), (e), and (f)).

3.2.2.3 Recalling the Previous Context Display

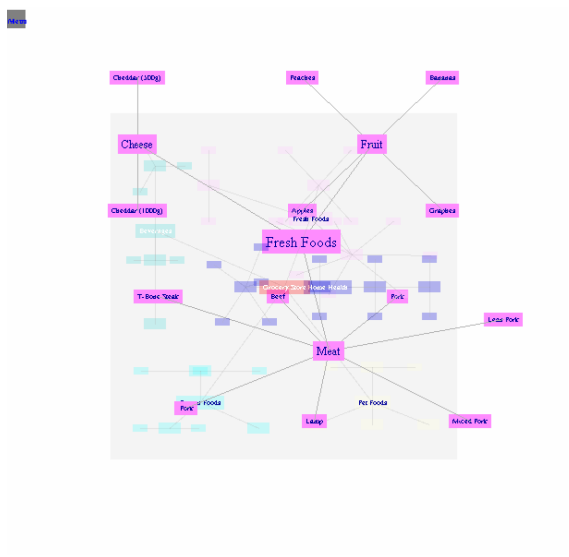
We also provide a reversed navigation mechanism to allow users to move back to the previous context views. This can be done through a right mouse-click. During the reverse navigation, the nodes in the history layer L_2 will change their colour back to the normal colour and increase their size gradually to the normal size as they use to be in L_1. The display area of the history hierarchy in L_2 also expands smoothly into the entire display area while the nodes displayed in L_1 fade into the background colour.


Menu
Tea
Beverages
Coffee
Chocolate
Cheese
Fruit
Fresh Foods
Meat
Bath
Grocery Store
Home Health
Garbage
Medicine
Sea Food
Frozen Foods
Milk Products
Other
Dog Food
Pet Foods
Fish Food
Cat Food
Birth Food

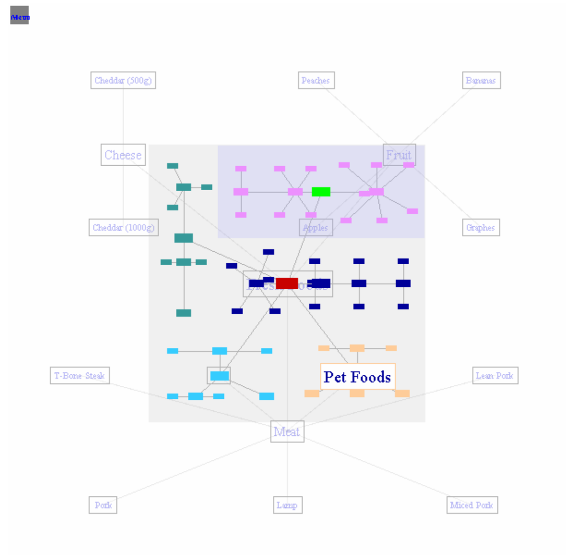
(a)


Menu
Cheddar (500g)
Peaches
Bananas
Cheese
Fruit
Cheddar (1000g)
Apples
Graphes
Fresh Foods
T-Bone Steak
Beef
Pork
Lean Pork
Meat
Pork
Lamp
Minced Pork

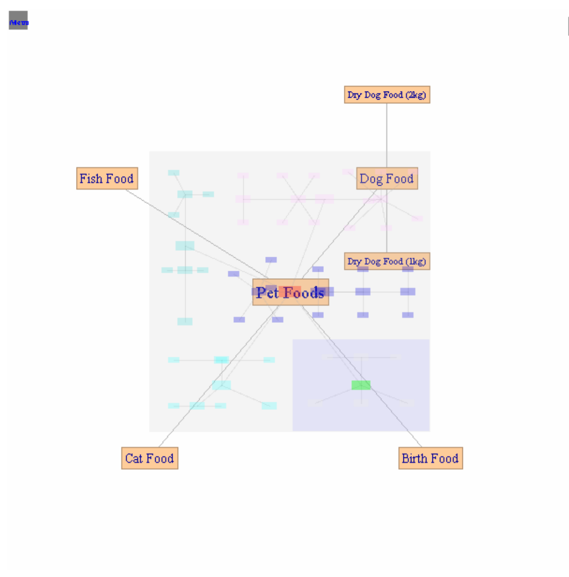
(b)


Menu
Cheddar (500g)
Peaches
Bananas
Cheese
Fruit
Cheddar (1000g)
Apples
Graphes
Fresh Foods
T-Bone Steak
Beef
Pork
Lean Pork
Meat
Pet Foods
Lamp
Minced Pork

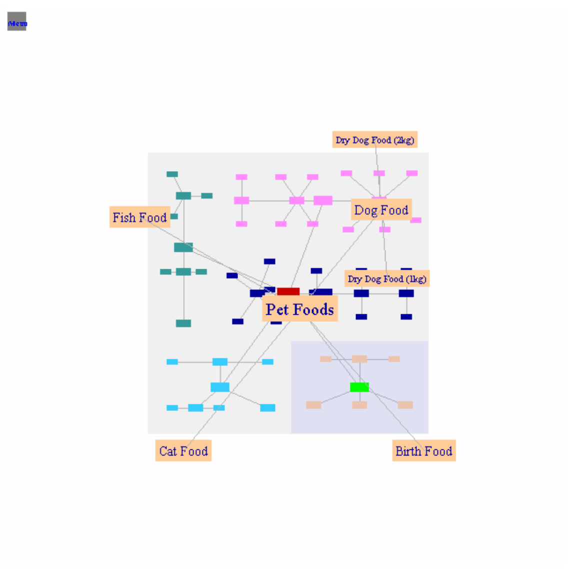
(c)


Menu
Cheddar (500g)
Peaches
Bananas
Cheese
Fruit
Cheddar (1000g)
Apples
Graphes
T-Bone Steak
Pet Foods
Lean Pork
Meat
Pork
Lamp
Minced Pork

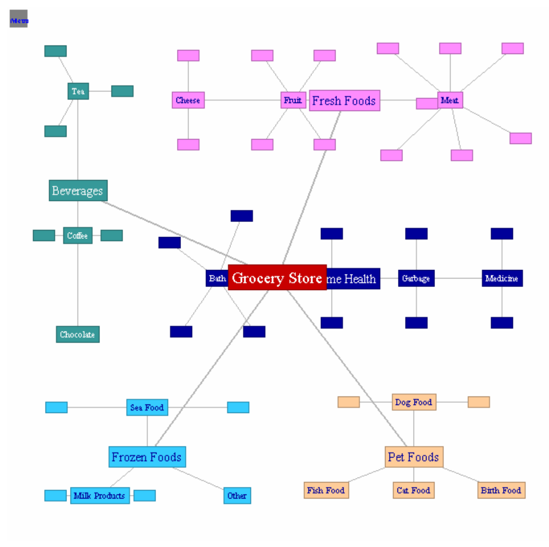
(d)



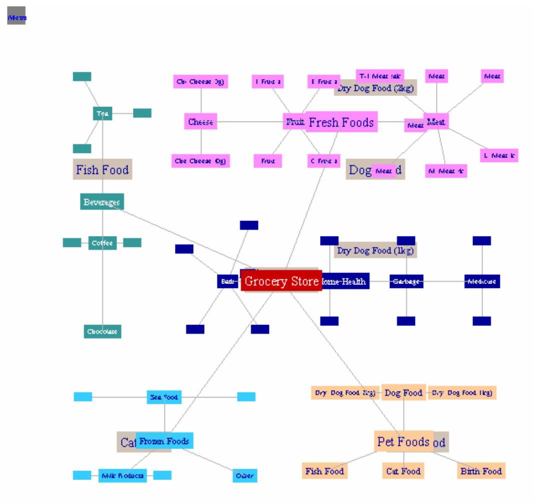
(e)



(f)



(g)



(h)

Figure 3.10. Examples of viewing, interactive navigation and animation in *layering display and interactive navigation*.

Figure 3.10 shows an example of the interactive navigation process through a small dataset. Image (a) shows the layout of the entire hierarchy before navigation. You can see that a user is focusing on a sub-hierarchy rooted at 'Fresh Foods' and is intending to navigate the detail of this sub-tree. Image (b) shows the display of after clicking on the focused node 'Fresh Foods'. This is at the 'default' mode of the display in which the detail view is highlighted in L_1 and the context view is displayed with lighter and semi-transparent colours in L_2. Image (c) illustrates an animated transition between images in (a) and (b). Image (d) presents the same visualisation as shown in image (b) after switching to the 'context' mode in which the context view is brought into L_1 and highlighted while the detail view is sent back to the L_2 with lighter and semi-transparent colours. We can also see that the user is now focusing on the node 'Pet Foods' and is intending to view the detail of this sub-hierarchy. Image (e) shows the visualisation after clicking on the focused node 'Pet Foods'. Image (f) illustrates an animated transition between images in (d) and (e). Image (g) is the visualisation of the entire hierarchy after a right click on the node 'Grocery Store' (reverse navigation). Finally, image (h) shows an animated transition between images in (e) and (g).

3.3 Summary

This chapter has presented the technical details of two new *focus+context* interactive navigation techniques called the *hybrid view,* and the *layering view.* These navigation techniques are independent to the layout algorithms. In short, the *hybrid view* method consists of two parts: the *browsing* and the *distortion*. The *browsing* is responsible for bringing a sub-hierarchy to the focus region while the *distortion* is used to enlarge information at the focus area. *Hybrid view* also employs *semantic zooming* to reduce the amount of information displayed in the focus view when the visualisation is overcrowded. On the other hand, the *layering view* uses the layering and semantic zooming to archive the *focus+context* navigation. Technically, the *layering view* employs semi-transparency to display concurrently both the *context* and the *focus* views at two separate layers. This technique also provides mechanisms to switch between views for the navigation. This allows users to easily navigate across the context layer or the focus layer.

The advantage of the *hybrid view* technique is that it speeds up the navigation process allowing a very large dataset to be quickly visited on a normal personal computer. The *semantic zooming* in *hybrid view* can quickly reduce the density of information by enlarging a focused sub-tree and removing the context information from the display. The employment of a simplified fisheye distortional technique also reduces the computational cost during the navigation. On the other hand, the *layering view* can display both the *focus view* and the *context view* on two separate layers of the same display medium by using the semi-transparent technique. This allows more information to be displayed in both the context view and the focus view because there is no area division required.

The selection of a particular navigation method depends on the property of a particular system that needs to be visualised. If the space utilisation is the main concern, then the *layering view* technique would be a better choice. Otherwise, the *hybrid view* might be preferred if the interactive speed is the main issue in the design of the visualisation.

Chapter 4 Case Study 1: A Visual Interface in Shared Collaborative Workspaces

Shared collaborative workspaces provide computer-based virtual environments supporting collaboration among organisational members. They allow organizational members to share documents or artefacts; to communicate through discussion forums, text chat and audio/video conferences; and to define and enact work processes. The use of shared workspace systems is increasingly becoming part of current work practice, particularly in large, distributed organisations where virtual teams form in order to jointly carry out collaborative project work.

One application area where there has been little research attention to the visual interface is that collaborative workspaces. Most of the current interfaces in collaborative workspaces display information via text-based interfaces. These presentations do not provide a meaningful visual representation of various types of the logical relationships among the objects which are involved in a collaborative learning environment. Thus, it becomes difficult for non-expert users to understand the logical relationships among objects in a working-window of the collaborative environment. Furthermore, the logical relationships among data objects across different working-windows are also almost impossible to be visualised using the traditional textual interface.

This chapter presents a visual interface for visualising and manipulating collaborative objects in a shared workspace, called *LiveNet.* Our new visualisation not only provides users with a Graphical User Interface (GUI) for viewing, browsing, analysing and manipulating collaborative objects, but it also visualises multiple relationships among these collaborative objects. The new visualisation consists of five main properties:

- a modified *EncCon tree* layout algorithm,
- a semi-transparent viewing technique,
- a clustered viewing technique,
- an animated *focus+context* interactive navigation technique,

- and a set of rich graphical properties, including a variety of icons, pop-up windows, shapes, colours, and others.

The combination of these properties aims to achieve the effective visualisation of multiple relational structures on the same display window. This visualisation is built into a GUI prototype that provides users with not only a two-dimensional graphical visual interface for direct data manipulation, but also a combined view of multiple relationships among the collaborative objects in shared workspaces.

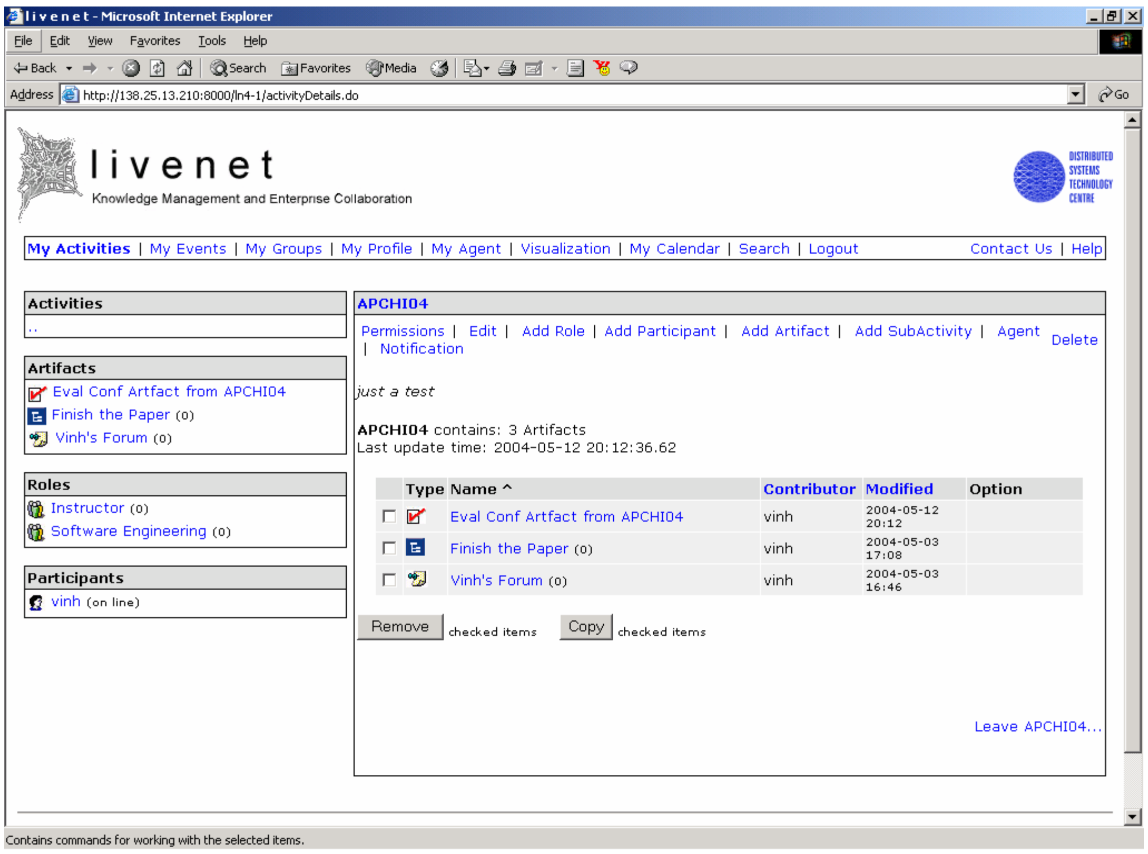
4.1 Shared Workspaces - LiveNet

Shared collaborative workspaces are defined as the use of computer-based virtual environments to support collaboration among organisational members. These workspaces allow organisational members to share documents or artefacts; to communicate through discussion forums, text chat and audio/video conferences; and to define and enact work processes. The use of shared workspace systems is increasingly part of current work practice, particularly in large, distributed organisations where virtual teams form in order to jointly carry out collaborative project work.

Collaboration through shared workspaces brings challenges in displaying the relation in multiple relationships, types and properties of collaborative objects through the user interfaces. The traditional text-oriented interfaces from shared workspaces in (Hawryszkiewycz 1999), (Appelt 1999), and (Roseman & Greenberg 1996) are seldom comprehensible to the user in terms of presenting relationships among collaborative elements. This costs extra cognitive overhead for users to understand the underlying structures and interrelations behind the datasets where the users are manipulating.

LiveNet (Hawryszkiewycz 1999) is one such shared workspace system developed in 1999, which provides the flexibility to customise workspaces by adding roles and artefacts, assigning permissions, adding actions and so on. LiveNet workspace is based on a meta-model that is stored as a relational database. This system is developed using Java 2 Enterprise Edition (J2EE) platform which provides the flexibility to both easily add new components as well as develop specialised interfaces. A development server of LiveNet collaborative workspace can be accessed at http://138.25.13.210:8000/ln4-1 (accessed 19/05/2004).

Although *LiveNet* is an excellent system for knowledge management and collaborative environment, its text-oriented interface is seldom comprehensible to the users. It costs extra cognitive overhead in understanding the underlying structures and interrelations behind the datasets that *LiveNet* is manipulating. For example, the current text-based interface in *LiveNet* does not provide a meaningful visual presentation of various types of the logical relationships among the objects which are involved in a collaborative learning environment. Thus, it becomes very difficult for non-expert users to understand the logical relationships among objects in a working-window of the collaborative environment as well as the logical relationships among data objects across different working-windows. In addition, the current user interfaces in shared workspace systems are not efficient for displaying large scale information because of the nature of the text-based interfaces. Figure 4.1 shows an example of the current text-based interface of the *LiveNet* system.


l i v e n e t - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://138.25.13.210:8000/ln4-1/activityDetails.do
Go
DISTRIBUTED SYSTEMS TECHNOLOGY CENTRE

l i v e n e t
Knowledge Management and Enterprise Collaboration

My Activities | My Events | My Groups | My Profile | My Agent | Visualization | My Calendar | Search | Logout   Contact Us | Help

Activities
..

Artifacts
Eval Conf Artfact from APCHI04
Finish the Paper (0)
Vinh's Forum (0)

Roles
Instructor (0)
Software Engineering (0)

Participants
vinh (on line)

APCHI04
Permissions | Edit | Add Role | Add Participant | Add Artifact | Add SubActivity | Agent | Notification   Delete

just a test

APCHI04 contains: 3 Artifacts
Last update time: 2004-05-12 20:12:36.62

	Type	Name ^	Contributor	Modified	Option
☐		Eval Conf Artfact from APCHI04	vinh	2004-05-12 20:12	
☐		Finish the Paper (0)	vinh	2004-05-03 17:08	
☐		Vinh's Forum (0)	vinh	2004-05-03 16:46	

Remove checked items Copy checked items

Leave APCHI04...

Contains commands for working with the selected items.

Figure 4.1. An example of the current text-based LiveNet's interface.

4.2 Related Works

The application of visual shared workspaces has still received little research attention. This is especially true for viewing multiple relationships among collaborative objects. Most existing shared workspace systems use traditional textual interfaces. (Roth et al. 1997) presented several systems for visualising and manipulating information in electronic workspaces including *Visage*, *SAGE* and *SDM*. Each system was designed to perform on a particular task. Particularly, *Visage* was used as user interfaces for multiple visualisation, analysis and communication applications. *SAGE* was designed as a tool for users to automatically and interactively create visualisations. Finally, *SDM* was a system for prototyping techniques for interacting with visualisations to modify their appearance (Roth et al. 1997). These systems, however, do not display simultaneously the multiple relationships among the workspaces' objects.

(Biuk-Aghai 1999, 2001) implemented a system to visualise the structural and behaviour aspects of virtual collaboration environments. This technique employed the force-directed animated visualisation algorithm to show the relational structures. The author also used colouring graphics to indicate the workspace's density. Similarly to (Roth et al. 1997), the logical relations among the information were not presented in its visualisation. This technique is not also able to visualise very large and complex collaboration environments due to the inefficient layout and navigation algorithms.

Visualisation of multiple relations has become one of the important topics in information visualisation research community. Although several available techniques are aimed to visualise simultaneously multiple relationships, few of them concern or use in visualising collaborative workspaces. We next review a few typical techniques in visualising multiple relations.

(Hao et al. 2003) described an approach to simultaneously layout in a graph the multiple relationships of web transactions. The idea of this technique is to freeze one set of objects before laying out the next set of objects during the construction of the graph. This technique was implemented in three-dimensional space and the relations were shown by freezing others. Although this technique visualise well the web transactions, it is not quite suitable for shared workspaces domains.

Another approach of visualising multiple relational structures (graphs) was proposed by (Erten et al. 2003). In the paper, the authors concerned more about theoretical perspective in drawing multiple graphs rather than the real world applications. This technique was based on a modification of the force-directed algorithm to visualise the multiple relational structures simultaneously. However, the visualisation proposed in the paper did not provide any mechanism for navigation through the structures and the use of slow force-directed layout algorithm might not be appropriate for handling large datasets.

The most common approach to visualise multiple relationships and/or multiple attributes of the data is to use multiple-views. It usually uses a sequence of small sub-windows to display the multiple views of the information. Each sub-window displays the data and data structures from one particular user's perspective. However, these techniques require the division of display space for presenting information. Thus, the available space left for displaying the information is getting even smaller. In addition, the users might be affected with distraction when switching their view from one window to the other window. More information about multiple-view techniques can be found at (Robert 2000), (Baldonado, Woodruff & Kuchinsky 2000), (C North & Shneiderman 2000, 2001), (Convertino et al. 2003), etc.

4.3 LiveNet's Visual Interface

To overcome the limitation of the current text-based interface and the above simple visual interfaces, as well as to explore internal structure of shared workspace, we present a new visualisation component that appears as an additional window embedded in *LiveNet* system. This visual interface can be used for viewing, learning, browsing, editing and manipulating collaborative information. This visual component employs *EncCon tree* layout algorithm and a semi-transparent navigation mechanism to handle large scale of learning information. The new visual interface aims to provide a better assistance to the users for visual manipulation and navigation of objects stored in a relational database. The use of visualisation techniques in the *LiveNet* provides the users with not only a two-dimensional graphical visual interface for direct data manipulation, but also views of the interrelations among the data objects which enhances the understanding of relational information that the users want to see.

In the design of visualisation component, the following objectives need to be achieved:

- Provide an interactive graphical interface in which users can view, learn, navigate and manipulate the entire information of the shared workspace.
- Improve the users' understanding of the underlying structures and internal relationships of the information. Thus, the new visual interface is able to visualise multiple relationships among collaborative objects in shared workspaces.
- Provide the overall context view of the entire workspace as well as a focused sub-workspace corresponding to a particular user.
- Be able to handle large amount of information.

Base on the original *LiveNet*'s system, its visual interface has been implemented using Java 2 Enterprise Edition (J2EE) technology to control the interaction between the client interfaces and the core system. With this dynamic data retrieval, the system is able to reflex the changes from the data source which makes it be valuable for collaborative environments. This applet window does not aim to replace entirely the traditional text-based interface, but it is as an extra assistance to the users.

This visualisation is built into a GUI prototype that provides users with 1) a two-dimensional graphical visual interface for direct data manipulation, 2) a combined view of multiple relationships among the collaborative objects, and 3) an attributed visualisation of collaborative objects and their relations in the shared workspaces. In order to display concurrently the visualisation of multiple relationships among the collaborative objects, we employ some advanced computer graphical techniques and animations to build a combined visualisation that can simultaneously display several relational structures (visual contexts) in the same window. This interface also provides the users an effective mechanism to switch between different visual contexts. The system is also able to display two or more visual contexts concurrently in the *LiveNet* shared workspaces. Finally, the system employs a numerous rich graphical elements to emphasize the property of all nodes and edges in the final drawing.

4.3.1 Relations in LiveNet Shared Workspaces

A database in a shared workspace system usually consists of multiple relations that are linked together conceptually via entity-relationship links in the design of the relational database schema. However, most information visualisation techniques only work on a single node-link representation of relational databases, in which the underlying data model is a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. It is usually difficult to represent multiple relational structures in a single visualisation, which usually only involves the drawing and displaying of one single graph G.

As mentioned above, *LiveNet* is a shared workspace containing several types of the collaborative objects, including *activities*, *artefacts*, *groups/roles*, *members*, and *action objects*. Note that the current *LiveNet* system does not identify the difference between a *role* and a *group.* Thus, a role can also be considered as a group. An object in the *LiveNet* can link to one or more objects with different types of relationships, including *categorising*, *accessing*, *participating*, *sharing*, and *occupying*. We now define some common relationships one by one below:

- *Categorising relationship* – this type of relationship is the classification relation that represents the nature ordering/grouping of the collaborative objects in the relational database. This hierarchical structure illustrates the relation among a *group* to a *subgroup*, an *activity* to a *sub-activity*, and any other type of parent-child relationships in the *LiveNet* system.
- *Accessing relationship* – this presents relationships among *documents* (or *artefacts*) and *activities.* The visual interface only displays the *documents* and *activities* which the current user has been authorised to access to.
- *Participating relationship* – this type of relationship shows the relation between an *activity* and a *role* (or a group). In other words, this relationship indicates the participation of role corresponding to an activity in the collaborative process.
- *Sharing relationship* – this is the relationship between two objects that share the same information, i.e. artefact's sharing. The sharing relationship is inherited when an artefact is copied and sent to another activity.
- *Occupying relationship* – the relationship represents the relation between a *role* and a user who occupies this *role* in the shared workspace. This relationship is recorded as a role's attribute from the user.

4.3.2 The Frame-Work of Visual User Interface

The *LiveNet* system was developed using Java 2 Enterprise Edition (J2EE) platform and was made of several components. Within the scope of this thesis, we concentrate only on the discussion of visual interface component and its associations from the shared workspace. Figure 4.2 shows the components and tiers of the visual user interface.

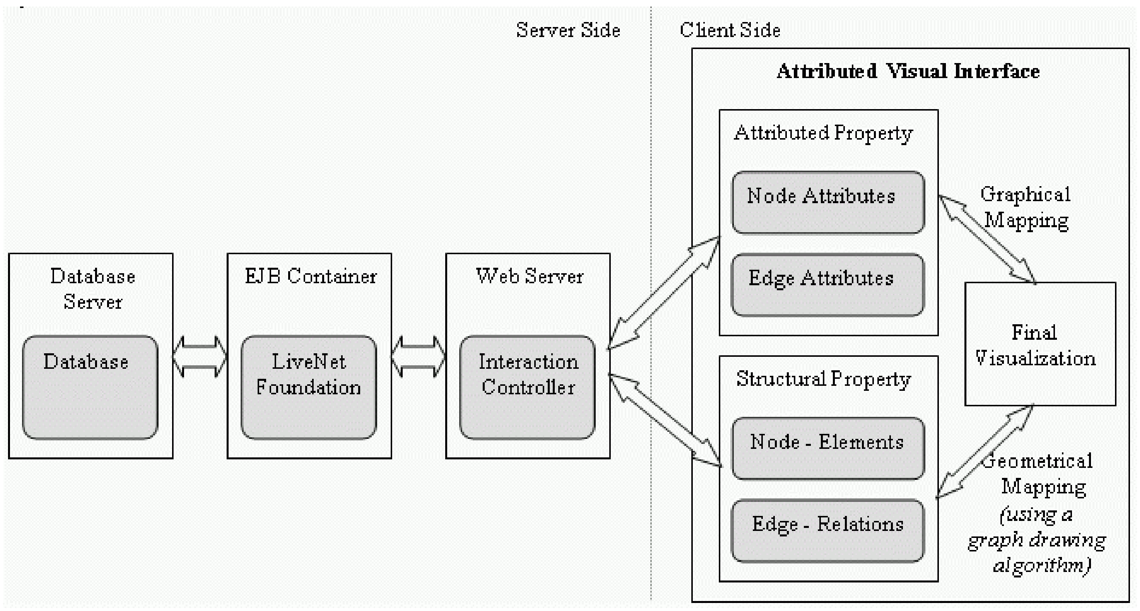


Figure 4.2. The framework of the visualisation system.

- *Attributed visual interface* – this is a visual navigational interface that automatically displays the entire workspace of a user. Objects in a shared workspace include activities, groups, artefacts, agent, events, and others. This interface is implemented using a Java Applet programming language and is intended to visualise not only the multiple relations of the entire collaborative workspaces but also a visualisation of attributes associated with collaborative objects and relations. From this interface, the users can also view, analyse, navigate, interact and manipulate collaborative objects in their workspaces. Detail of all components in the *attributed visual interface* is described below.
    - *Structural property* – this component is a set of nodes and edges which represents the relational structure of the entire collaborative workspace. Formally, it can be defined as a graph $G = (N, E)$ consisting a finite set N of

nodes and a finite set *E* of edges. A node represents a collaborative element and an edge represents a relation of two objects.

  - *Attributed property* – this component is a set of domain-specific attributes or properties that are associated with collaborative elements and/or relations. There are two types of attributes: node attributes and edge attributes which represents the properties of collaborative elements and relationships among objects in the workspaces. The attributes of the nodes and edges are dynamically retrieved and updated from the server-side to client-side and vice versa.
  - *Final visualisation* – this is the final attributed representation of the collaboration workspaces. This includes the representation of collaborative elements, relational structure and the attributes that are associated with a node and/or an edge. The system first produces a geometrical layout of the entire collaborative workspaces using *EncCon tree* layout algorithm. The system next applies the graphical properties, such as transparency, shape, size, colour, brightness, edge thickness, etc, to represent concurrently multiple relations as well as the attributes of nodes and edges.

- *Interaction controller* – this component is the web server tier which is responsible for creating the initial visual interface, as well as receiving requests, executing, and sending responds to the visual interface component. This component is implemented using a Java Servlet programming language.
- *LiveNet foundation* – this core component contains a collection of classes and/or functions that perform the *LiveNet* functionalities as well as the database connections.
- *Databases* – these are relational databases where all information (collaborative objects and relations) of the *LiveNet* system is stored.

4.3.3 Combined Visualisation

Several techniques are employed in the visual interface component which aims to satisfy the following expectations:

- Provide the concurrent visualisation of multiple relationships among collaborative objects within the users' workspaces.
- Provide an effective interactive visual interface from where users can interactively view, navigate and manipulate objects in the entire workspaces.
- Provide an effective visualisation of attributes associated with collaborative objects and the relationships among those objects.
- Be able to handle large amount of information.

In our visualisation system, a node represents a collaborative object, while an edge represents a relationship between two objects. The geometrical layout of objects is based on either their *categorising* relation or *accessing* relation. This ensures the order of structural hierarchy of the entire workspace as well as a particular section of the workspace. Figure 4.3 shows an example of the entire workspace corresponding to a particular user. This visualisation aims to provide 1) a two-dimensional graphical visual interface for direct data manipulation, 2) a combined view of multiple relationships among the collaborative objects, and 3) an attributed visualisation of collaborative objects and their relations in the shared workspaces which enhances the understanding of the collaborative workspaces.

EncCon tree layout algorithm was used to present the hierarchical structure of the collaborative workspaces. This algorithm was chosen for our implementation in order to take advantage of its efficient utilisation of geometrical space, fast calculation, and aesthetic quality. Figure 2.18 to Figure 2.21 are examples of the visualisation using the *EncCon tree* layout algorithm.

We visualise concurrently all kinds of relationships among collaborative objects within users' workspaces in a single display window. As mentioned above, the types of relations in *LiveNet* system include: *categorising*, *accessing*, *participating,* and *sharing*. Several techniques are provided to support the concurrent display of multiple relationships. The visualisation only allows one or two certain types of relations to be actively shown at a time while the other relations are deemphasised using lighter and semi-transparent colours. This property aims to solve the problem of distraction caused by many types of the relations, yet the users can still see all of the relationships. Further description of the concurrent visualisation of multiple relations using semi-transparency is presented in section 4.3.3.1.

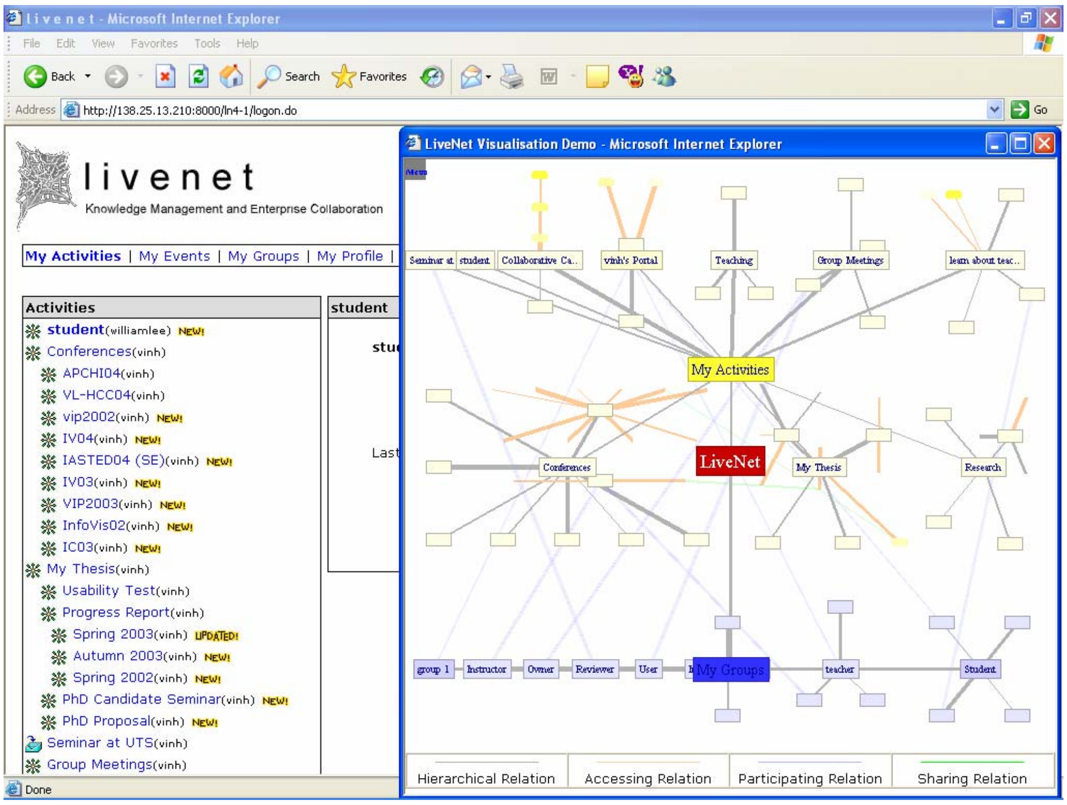


Figure 4.3. An example of the visualisation of a particular user's shared workspace.

The visual interface employs several graphical techniques, such as colours, brightness, shapes, sizes, icon styles, edge thickness, etc to visualise the attributed properties of all elements and relations in the workspaces. This visualisation provides not only a clear structural view of the entire workspace but also a view of associated attributes of all collaborative objects and their relations in a single display window. As a result, the users can easily identify the types and properties associated with each collaborative element as well as its relationships. More description of attributed visualisation of the collaborative workspaces is shown at section 4.3.3.2.

We use *layering view* navigation in collaborating with the fade in/out animation and semi-transparent viewing to support fast navigation of large graphs. In this navigation, the focused sub-graph is displayed as the *detail-view* and the context is displayed as the *global-view* in a semi-transparent manner. This allows users to explore through entire workspace quickly by moving around the structure by switching between the detail-view and the global view. We also use the pop-up windows to identify the occupying relationship when the mouse is over a node. Further description of the interactive navigation method is given at section 4.3.3.3.

4.3.3.1 Concurrent Visualisation of Multiple Relationships

As mentioned at section 4.3.1, there are several types of relationships existing among the collaborative objects, namely *categorising*, *accessing*, *participating*, *sharing*, and *occupying*. To be able to concurrently display these multiple relationships in a single visualisation, three graphical techniques are employed including colouring, semi-transparency, and clustered viewing.

In this system, each type of relation is drawn with a different colour in order to make the relation stand out from other types of relations. The default colours of *categorising*, *accessing*, *participating*, and *sharing* relations are respectively grey, sandy-brown, light-blue and green. These colours can be adjusted by the users. The *occupying* relation is associated with each activity or group/role object. Thus, we do not use edges to show this type of relationships rather we use pop-up windows.

In the visualisation, each type of relation displayed in the screen must be in either the *active* or *inactive* state. The *active* relations are drawn using real colours while the *inactive* relations are drawn with lighter, semi-transparent colours. Figure 4.4 shows an example of the display when the *categorising* and the *accessing* relations are at the active state, and the *participating* and the *sharing* relations are at the inactive state. Figure 4.5 shows the display of the same dataset when the *participating* and *sharing* relations are switched to the active state, and the *categorising* and the *accessing* relations are at the inactive state.

In order to reduce the complexity of the visualisation, at any time, only two types of relations at most are shown with the active state while the other types of relations are shown at the inactive state. A particular user can interactively change the state of each type of the relations through the option menu, to switch to a specific visual context he/she wants.

We also provide a clustered viewing approach to enhance the clarity of the visualisation by reducing the complexity of the display of logical edges. Users can interactively switch between the *normal view* and *clustered view*. At the *clustered view*, if there are more than two links from a node to a sub-graph, i.e. a group participates in an activity as well as its sub-activities, and then a thick edge is drawn to replace a set of edges showing high level abstracted linkage information. The other edges are drawn as thin lines.

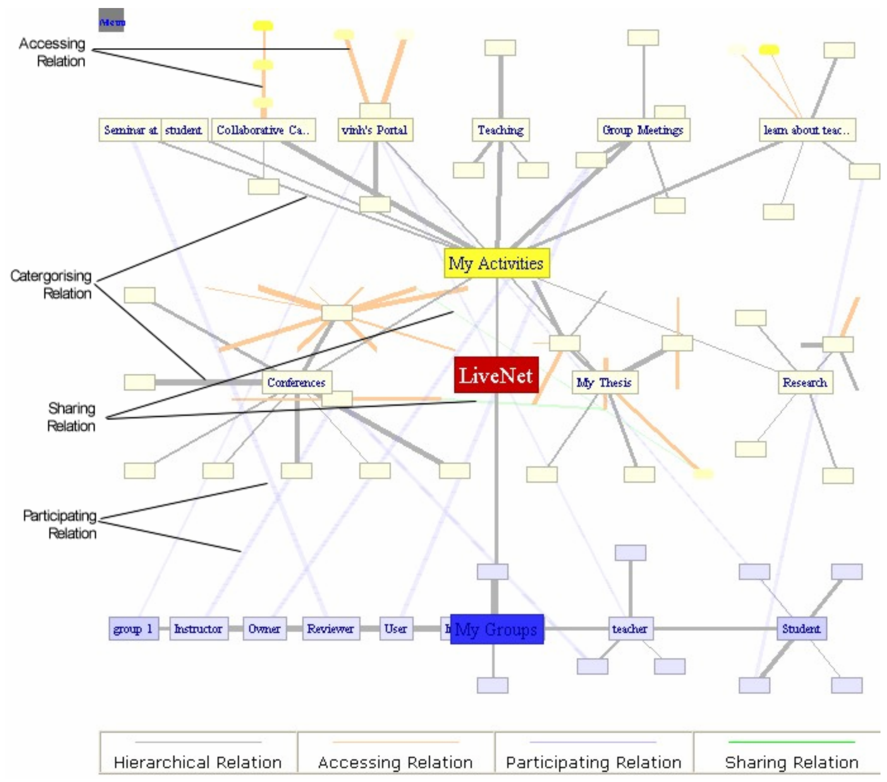


Figure 4.4. An example of the display where the *categorising* and the *accessing* relations are at the active state.

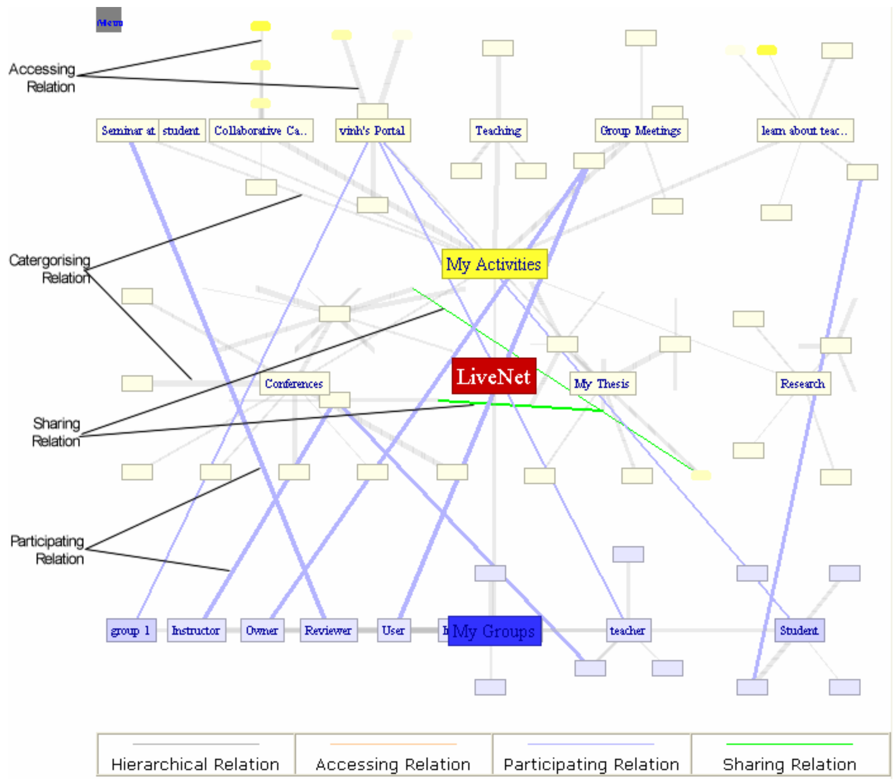


Figure 4.5. An example of the display where the *participating* and the *sharing* relations are at the active state.

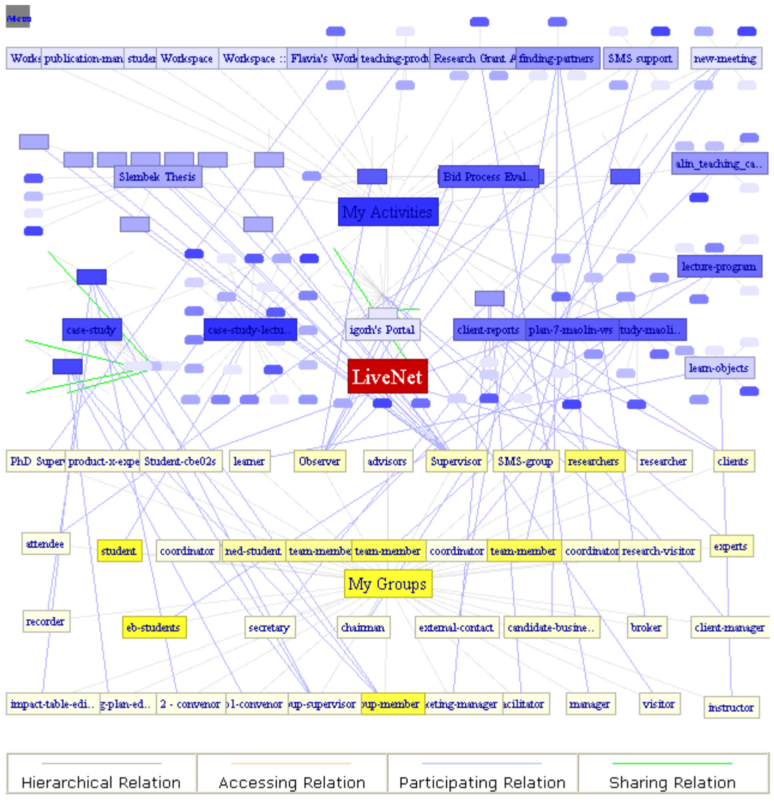


Figure 4.6. An example of the display or a workspace with a normal view.

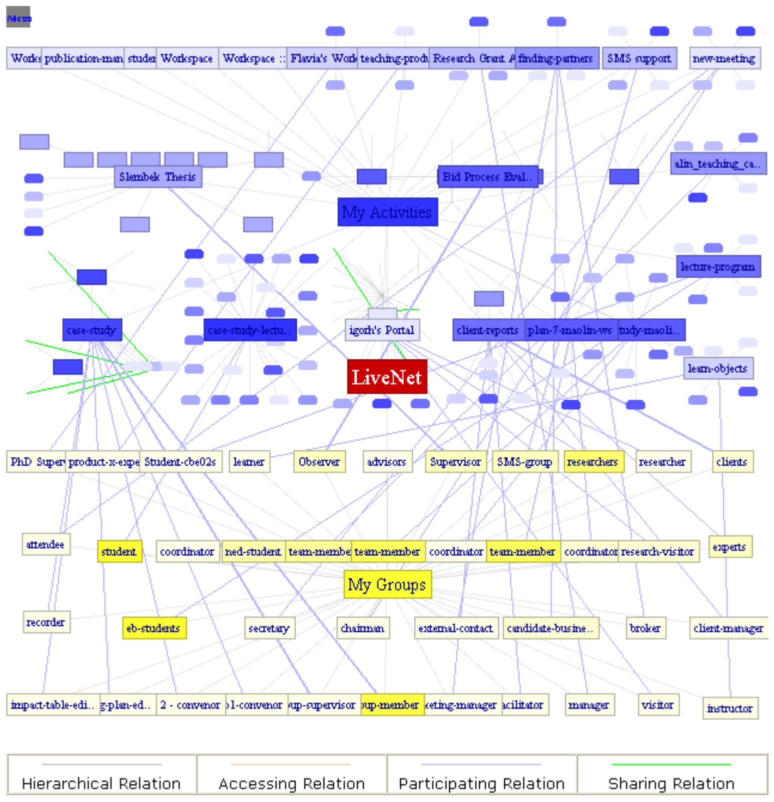


Figure 4.7. An example of the display of a workspace with a clustered view.

Figure 4.6 shows a workspace with the normal view. Figure 4.7 shows the same workspace with a clustered view with the relations from nodes ‘Client’, ‘Supervisor’ and ‘Observer’, etc, clustered and drawn as thick edges. Obviously, Figure 4.7 gives a much clearer view than what Figure 4.6 does. This is because many edges are filtered from the visualisation shown in Figure 4.6.

4.3.3.2 Attributed Visualisation

In this application, we employed several graphical attributes to visualise the domain-specific attributes associated with the collaborative elements and relations in the workspaces. We next describe details of attributed properties of the collaborative entities as well as their relations.

Entity attributes - an entity or a collaboration object and its domain-specific properties are represented as a node with attributed graphics in our visualisation. The attributed mapping between the collaborative workspaces and the visualisation is:

- Node-size represents the hierarchical level or the depth of a node in the hierarchy. In our visualisation, the size of a node at a deeper level is smaller than a node at a higher level (see Figure 4.4, Figure 4.8 and Figure 4.10).
- Node-colour represents a particular group of collaborative objects. There are two major groups of objects in *LiveNet* workspaces. Group-one includes all activities, sub-activities and artefact objects and group-two includes member groups in the workspaces. The colours of group 1 and group 2 are respectively yellow and blue (see Figure 4.8).
- Node-brightness represents the degree of importance which is roughly called a weight of an element. In our implementation, a node with darker colour is more weight than a brighter node. Specifically, the weight of an object is defined as below:
    - If a node is a group object, the weight represents the number of members of the group.
    - If a node is an activity or a sub-activity, the weight represents the number of participants involved in the activity or sub-activity.
    - If a node is an artefact, the weight represents a property of the artefact corresponding to each type of artefact. In short, if the artefact is a forum or a

chat-room, the weight is calculated based on the scale, i.e. numbers of messages in the forum or the chat-room; or if the artefact is an uploading file, a view folder or a text message, the weight is calculated based on the size of the file, folder and message, and so on.

- Node-icon and/or node-shape are used to represent the type of a collaborative object. The iconic visualisation can slightly reduce the clarity of node-colour and node-brightness compared to the other style. This visualisation, however, improves perception significantly when a user analyses the types of collaborative objects in the workspace (see Figure 4.10). In our system, the user can easily switch the display between the iconic mode and the rich-graphics mode. In the rich-graphics mode, an artefact is drawn as rounded-rectangle while an activity or a sub-activity is drawn as a normal rectangle. This simple node-shape style allows the user to quickly clarify the differences between two types of objects (see Figure 4.9).

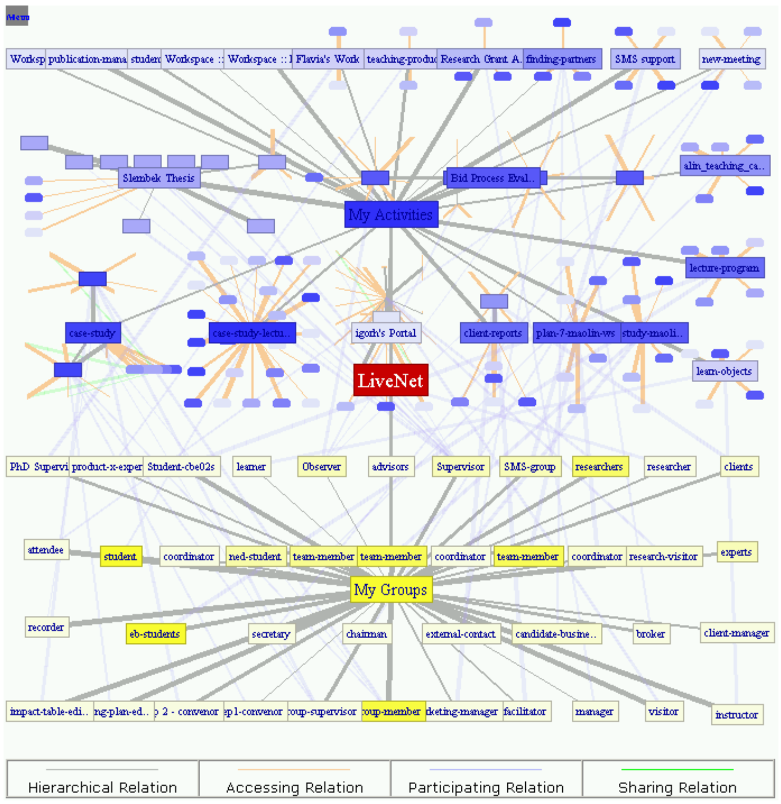


Figure 4.8. An example of the attributed visualisation of a large workspace.

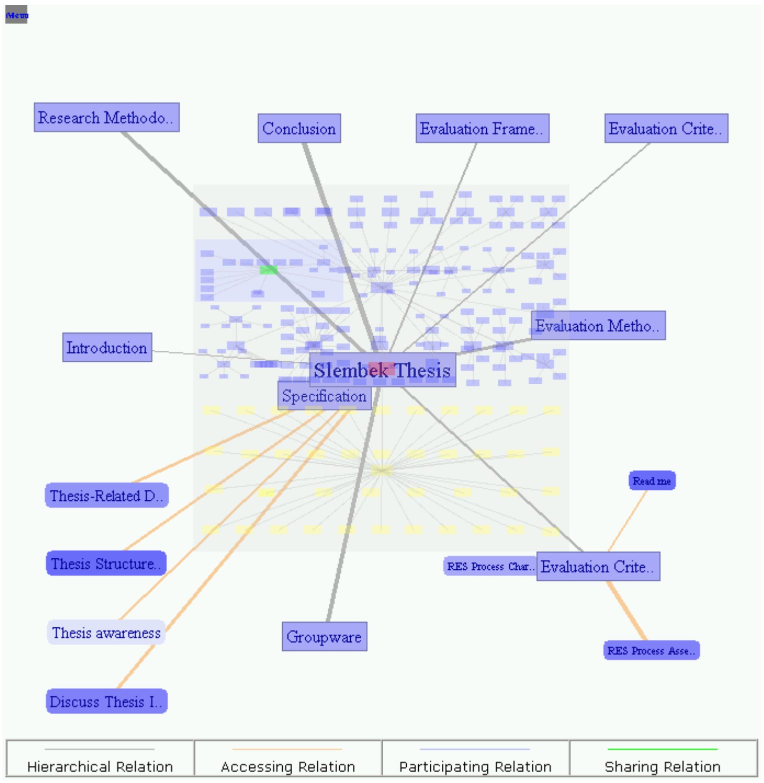


Figure 4.9. An example of a detail look from the attributed visualisation when a subsection is enlarged during the navigation.

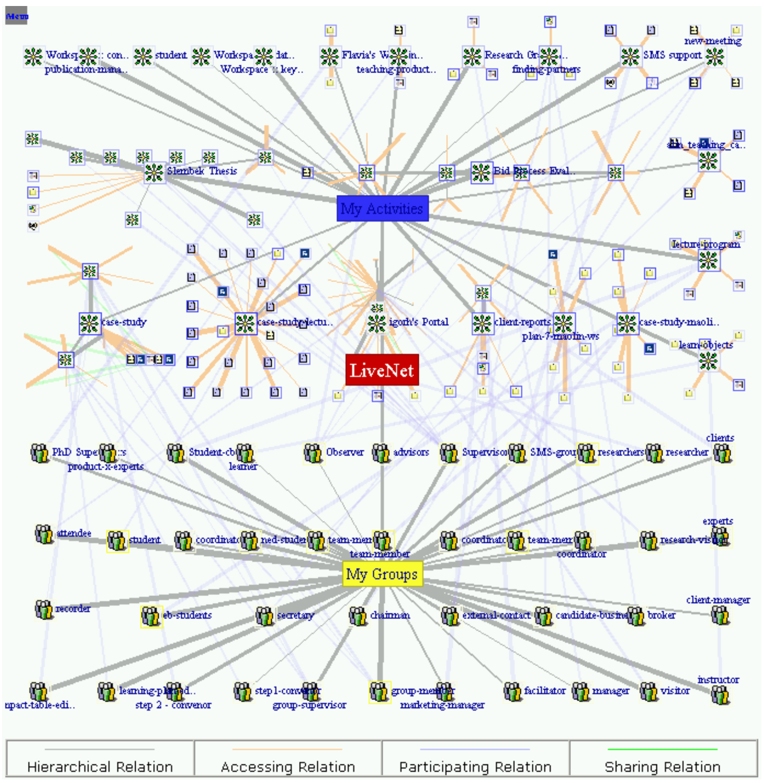


Figure 4.10. An example of the attributed visualisation when nodes are represented as icons corresponding to their types.

Figure 4.8 shows an example of the overview of the entire workspace that the important degree of each collaborative object is shown on each node. Figure 4.9 is another example of the structural detail of a focused activity in the workspace. In short, this figure shows clearly that the activity 'Slembek' Thesis" has several sub-activities and four artefacts. As can be easily seen, the artefact 'Thesis Structure' has the most weight while the artefact 'Thesis Awareness' has the least weight. Figure 4.10 illustrates the visualisation when icons are used to represent the types of collaborative objects. This figure indicates that the system currently has only one type *group* object, one type *activity* object, and several types of *artefacts*.

4.3.3.3 Interactive Navigation Technique

As mentioned above, we use the *layering view* approach with fade animation and semi-transparent viewing to support fast navigation of large workspaces. The semi-transparent viewing technique is used to support fast navigation of large graphs in which a focused sub-graph is displayed as the detail-view in a full display and the context of the sub-graph is displayed as the global-view with reduced size appearing in a semi-transparent presentation. These two views can be toggled smoothly through the fade in/out animation interactively. This allows users to explore through entire workspace quickly by moving around the sub-graphs. Each visual interaction is accommodated by a fade animation in order to preserve user's cognitive-map of views during the navigation.

More information about how to implement semi-transparency and fade animation techniques for achieving the *layering view* navigation can be found in section 3.2. Figure 4.11 shows the display of the entire workspace and the user is currently focusing on the node 'My Activities'. Figure 4.12 shows an example of the navigation using semi-transparency when the node 'My Activities' is selected from Figure 4.11. You can see from the display of Figure 4.12 that the content of the node 'My Activities' is enlarged to occupy the entire screen, while the previous context view in Figure 4.11 is reduced and sent to the background with semi-transparency. Animation is used to preserve the user's mental maps for the transitions from Figure 4.11 to Figure 4.12 and vice versa. Technically, the detail view is enlarged smoothly through the fade animation to the entire display area, while the context view is reduced and sent back to the background of the window at the centre, and the current focused section is bolded at the context view.

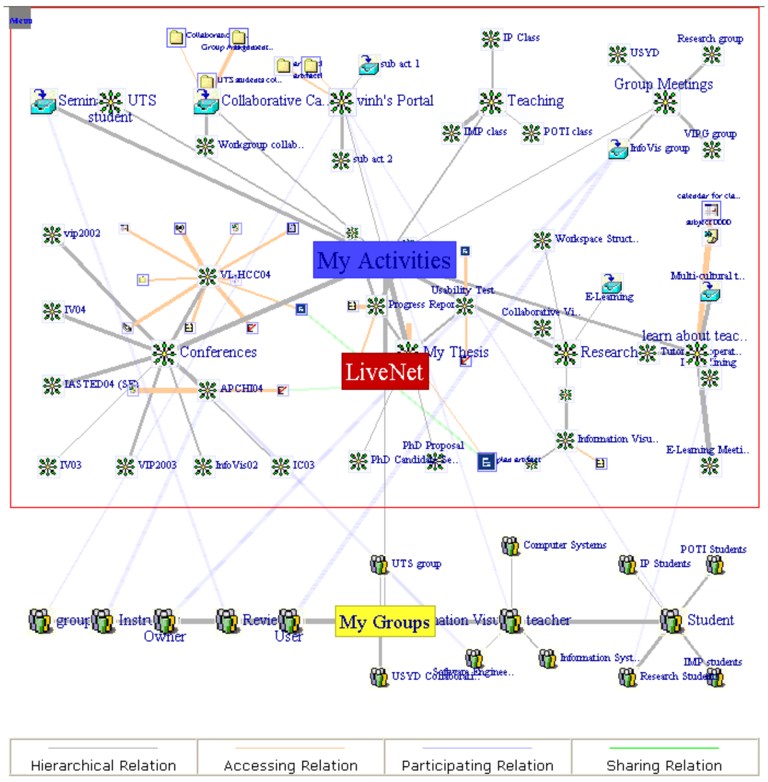


Figure 4.11. An example of the display when the view at Figure 4.3 is switched to iconic mode.

This image also indicates that the node 'My Activities' is being selected.

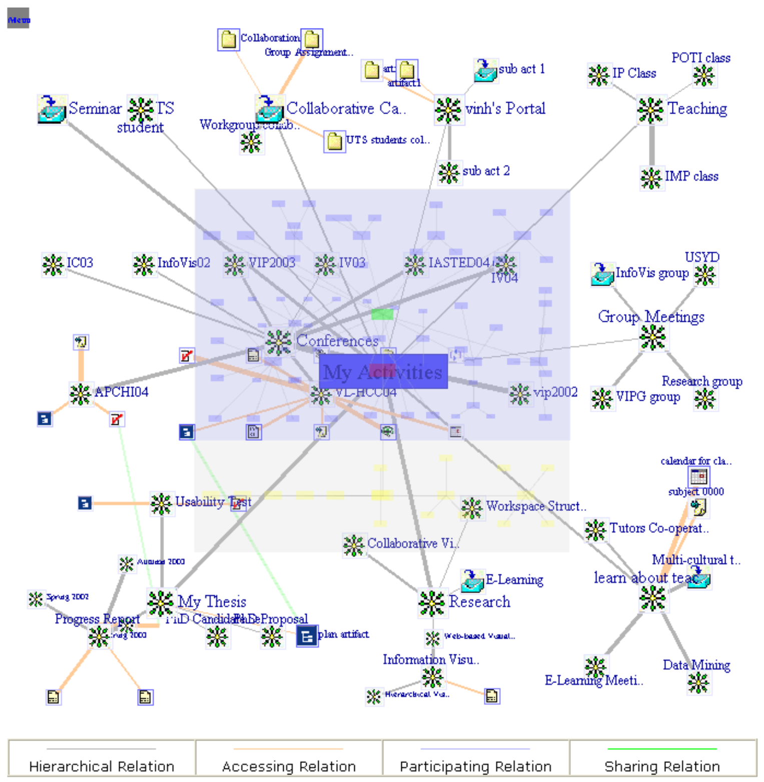


Figure 4.12. An example of the display when the node 'My Activities' is selected.

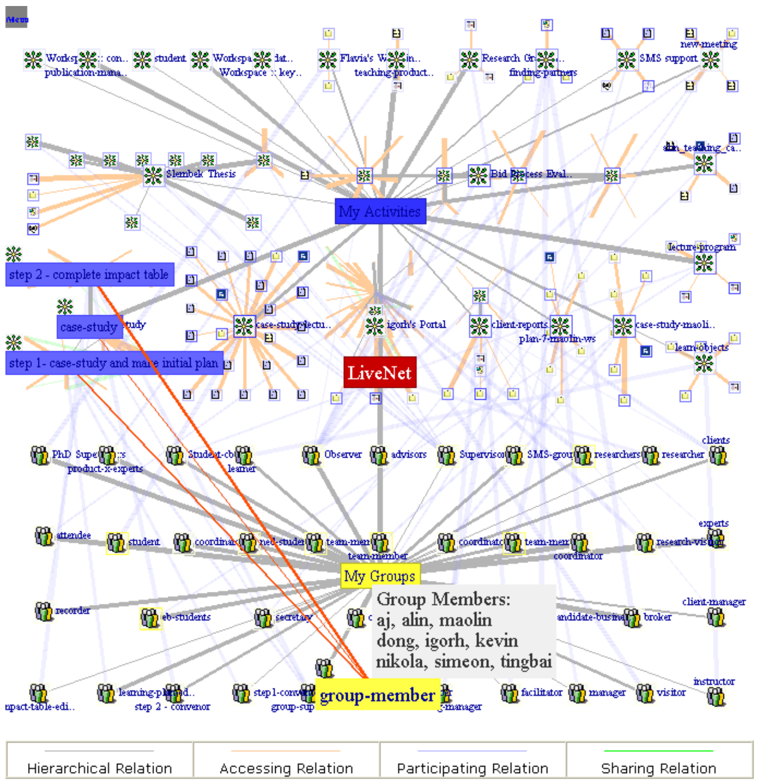


Figure 4.13. An example of the interaction when the user is pointing the mouse over a node.

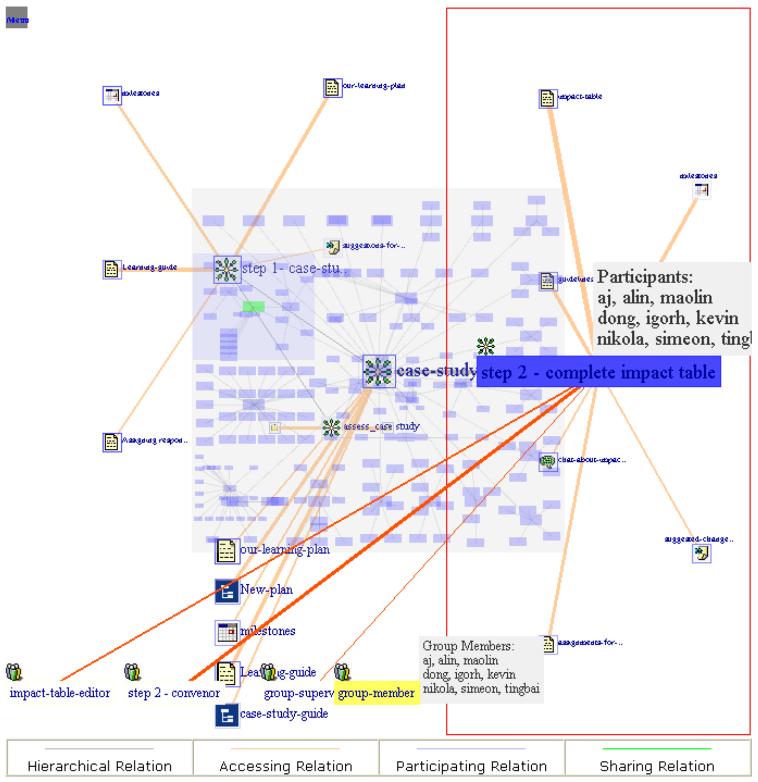


Figure 4.14. An example of the interaction when the view is held and the user can interact with more information through linked nodes.

We also provide interactive techniques to highlight both focused object and relations (including *participating* and *sharing* relations), as well as display the *occupying* relations. The *occupying* relationships indicate the members of a group (or a role) or participants of an activity. Figure 4.13 shows a screen dump of the visualisation that when the user rollovers the mouse to a group node 'group-member', we can see that a pop-up window is appeared that shows all the members belonging to the group. We can see also from the figure that all relationships linked to/from this focused group are activated and highlighted (including the activity 'case-study' and its two sub- activities 'step 1 – case study and make initial plan' and 'step 2 – complete impact table'). Those edges and nodes that are directly linked with the focus node are highlighted using respectively the red colour and light yellow. While a node is being focused, the user can press the control-key to hold the view (with highlighted linkage information) in order to further view, interact and navigate from the highlighted edges and nodes. The holding can be released when the control-key is pressed again. Figure 4.14 shows that the node 'Organizational Reference Material' is focused and its corresponding highlighted view is hold so that the user can release the mouse for further viewing and interacting with the 'Observer' group.

4.4 Summary

This chapter has presented technical details of a visualisation component as an additional window in shared collaborative workspaces. This component employs *EncCon tree* algorithm and the *layering view* techniques allowing users to view, browse, edit and manipulate collaborative objects/relations in *LiveNet* system. The *EncCon tree* is responsible for handling the geometrical layout of large hierarchical information. The system is able to visualise multiple relationships among collaborative objects. In particular five types of relationships, including *categorising*, *accessing*, *participating*, *sharing* and *occupying*, are concurrently displayed in a single visualisation by using *semi-transparency, clustered viewing* and *pop-up windows*. The visual interface also employs several graphical properties, such as *colours, brightness, shapes, sizes, icon styles, edge thickness, and others* to visualise the domain-specific attributes of all collaborative objects and relations in the workspaces. Therefore, in comparison with the traditional interfaces of collaborative systems, this visualisation component provides users with better assistance for viewing, navigating, understanding, analysing and manipulating of the collaborative elements in shared workspaces. The *layering view* navigation used in this application provides the capability of enlarged display of both the *context view* and the *focus view,* allowing more collaborative elements to be displayed in the *focus view*.

Chapter 5 Case Study 2: A Visual Browser for Large-Scale Online Auctions

The fast growth of the Internet has dramatically brought together more and more buyers and sellers in *electronic marketplaces*. Despite the rapid growth of interest and research into internet-based online e-commerce systems, the design of efficient mechanisms for navigating online product catalogues is still quite limited, especially for online auctions. This chapter describes an interactive visual interface for navigating product catalogues of large online auction sites. The *EncCon tree* layout algorithm was used to display the entire, as well as a portion of product hierarchy. The display of the hierarchy is accommodated with our new *layering view* technique for moving the focus point of users around the product hierarchy. An online auction prototype was developed to simulate the ordinary auction activities with the assistance of a proposed visual interface.

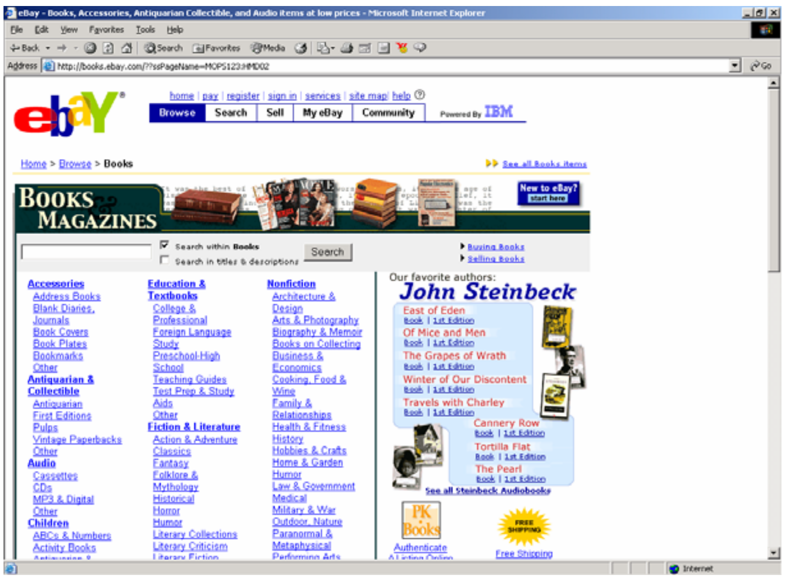
5.1 Product Catalogue's Navigation in Online Auctions

Over the past few years, electronic commerce (or e-commerce) has emerged as a dramatic new model of business (Bakos 1998). One of the greatest potentials of e-commerce is its ability to bring the effectiveness and unprecedented massive scale of buyers and sellers from all over the world. This property benefits both sides so that the buyers might have greater product diversity with potentially lower prices, and the sellers are able to reach a greater numbers of potential customers (Hahn 2001). At any time, through the online shopping stores (or auction websites), customers can learn more about the products, buy goods with electronic cash, and even have information goods delivered over the network. On the other hands, suppliers can reduce the overhead costs by investigating less in physical stores and distribution channels (Kim 1999).

An important precondition to the success of e-commerce systems, or specifically online auctions, is the construction of appropriate customer interfaces, from which online product catalogues can be retrieved, is one of the key elements. Many extensive research projects have been done on both components of the online product catalogue including the content management and the catalogue interface. For content management, a number of products have been developed and used at commercial website such as *CardoNet*, *Interwoven*, *OnDisplay*, *Poet Software*, *Vignette*, etc (Neumann 2000). Various methods that support product search and navigation have been developed for catalogue interface such as those systems in (Callahan & Koenemann 2000), and (Huang & Zhang 2002).

Currently, the majority of commercial auction websites provide users with both the basic click-through navigation scheme, which is based on HTML pages, tables and lists, and add-on navigation aids. The add-on navigation aids aim to provide navigation functions customisable to each user's need, such as search engines, and personalised recommendations. In addition, multiple views of lists are usually used for the ease of seeking interesting items. These views include 'Current' (i.e. the default view of all items), 'New Today' (i.e. the new items posted today), and 'Ending Today' (i.e. the items ending today), etc.

Although the available navigation techniques can effectively assist sellers/buyers in searching and accessing product information over the *World Wide Web*, they mainly use the text-based interface that allows users to navigate by clicking-through several pages via URL links. Thus, it could be difficult for the users to perceive the overall structure of the product hierarchy by reading these textural lists. Figure 5.1 shows an example of a text-based interface that is used on eBay's online auction website for browsing the product catalogue.


eBay - Books, Accessories, Antiquarian Collectible, and Audio items at low prices - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://books.ebay.com/??ssPageName=MOPS123:HMD02
Go
home | pay | register | sign in | services | site map | help
Browse Search Sell My eBay Community
Powered By IBM
Internet

Home > Browse > Books

See all Books items

BOOKS & MAGAZINES

New to eBay? start here

[✓] Search within Books
[ ] Search in titles & descriptions
Search

Buying Books
Selling Books

Accessories
Address Books
Blank Diaries, Journals
Book Covers
Book Plates
Bookmarks
Other
Antiquarian & Collectible
Antiquarian
First Editions
Pulps
Vintage Paperbacks
Other
Audio
Cassettes
CDs
MP3 & Digital
Other
Children
ABCs & Numbers
Activity Books

Education & Textbooks
College & Professional
Foreign Language Study
Preschool-High School
Teaching Guides
Test Prep & Study Aids
Other
Fiction & Literature
Action & Adventure
Classics
Fantasy
Folklore & Mythology
Historical
Horror
Humor
Literary Collections
Literary Criticism
Literary Fiction

Nonfiction
Architecture & Design
Arts & Photography
Biography & Memoir
Books on Collecting
Business & Economics
Cooking, Food & Wine
Family & Relationships
Health & Fitness
History
Hobbies & Crafts
Home & Garden
Humor
Law & Government
Medical
Military & War
Outdoor, Nature
Paranormal & Metaphysical
Performing Arts

Our favorite authors:
John Steinbeck
East of Eden
Book | 1st Edition
Of Mice and Men
Book | 1st Edition
The Grapes of Wrath
Book | 1st Edition
Winter of Our Discontent
Book | 1st Edition
Travels with Charley
Book | 1st Edition
Cannery Row
Book | 1st Edition
Tortilla Flat
Book | 1st Edition
The Pearl
Book | 1st Edition
See all Steinbeck Audiobooks

PK Books
Authenticate A Listing Online
FREE SHIPPING
Free Shipping

Figure 5.1. An example of the traditional text-based interface for online auction. Source from: (http://www.ebay.com, accessed 20/03/2004).

Some newly developed visualisation approaches have been proposed and implemented to enhance the presentation of product hierarchies for navigation. They aim to improve the readability, understandability, and comprehension of underlying hierarchical structure and to reduce the cognitive overhead for understanding the structure. These techniques are primarily for two-dimensional graph/tree visualisation techniques to display and navigate the product catalogues. The technical detail of these visualisation techniques can be found at (Huang & Zhang 2002), (Lee, Lee & Wang 2001), and (Inxight). However, there is still little research on the visual navigation of online auctions.

This chapter describes a new visualisation approach for navigating large-scale online product catalogues of online auction stores. The visualisation technique uses *EncCon tree* layout algorithm that displays the entire product hierarchy as well as a small portion of focused sub-hierarchy. Users can browse through the entire product catalogue via *layering view* technique. A prototype was developed to demonstrate the effectiveness of this visualisation technique in the area of online auction.

5.2 The Framework of Visual Online Auction Store

The proposed *visual online auction store* consists of several components. Within the scope of this thesis, we consider only the display and navigation components of the online auction. Figure 5.2 shows the components and interconnections among them in the context of online auction.

- **Product database** is a relational database used to store product information, including all data fields, attributes, and bidding information associated with a particular product that is available for auctioning. We used a MySQL database in our implementation.
- **Product Catalogue** is a content management system that assembles, indexes, aggregates and normalises product information from the product database, and quickly distributes the product catalogue information.

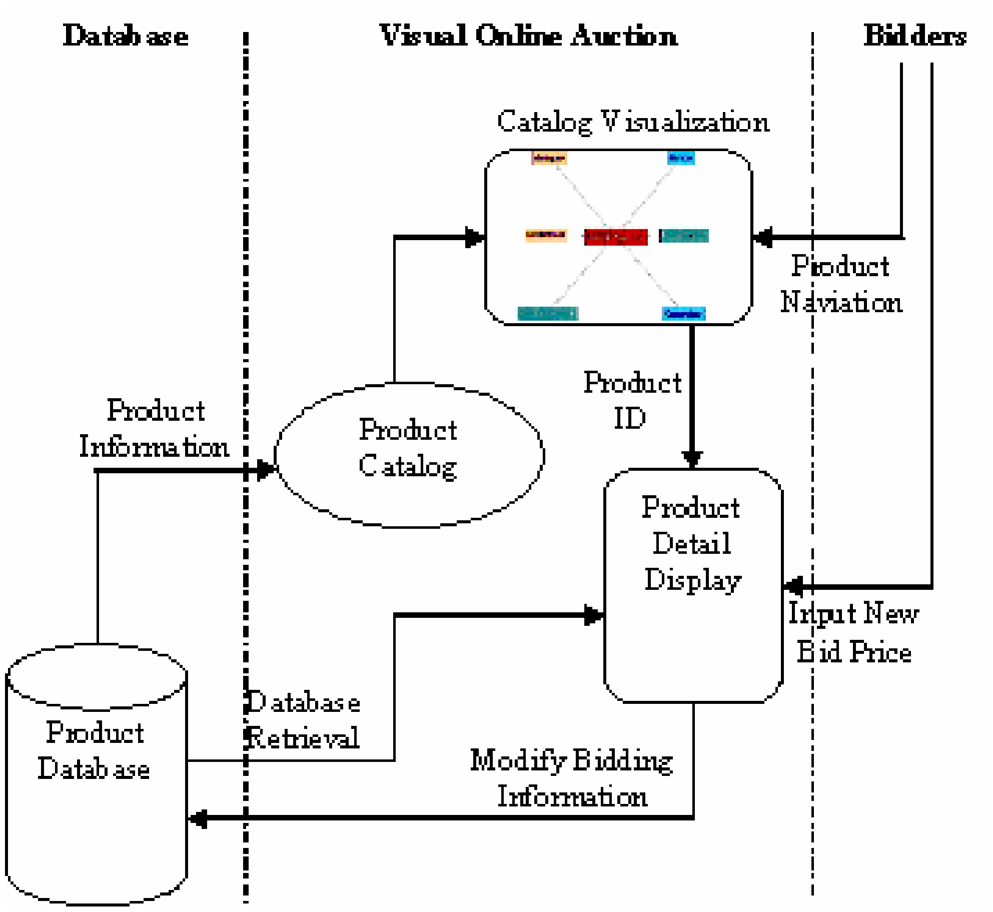


Figure 5.2. The framework of a visual online auction store.

- **Catalogue Visualisation** is a visual navigational interface that automatically displays the entire product catalogue's hierarchy, including categories, subcategories, and products. This component employs the *focus+context* visual layout and navigation mechanism from sections of respectively 2.4 and 3.2 that allows users not only to view the entire product hierarchy, but also to interactively browse down to a particular auctioned item.

- **Product Detail Display** is a web page generated on the server side by a particular scripting language, PHP, in our implementation, to show all the appropriate information of the selected product. This page also displays the product's bidding information, and it allows the authenticated bidder to input the bidding price for the product.

5.3 Dynamic Visualisation of Online Auction's Product Catalogue

The visualisation of the product catalogue for online auction is implemented using the Java programming language. The applet retrieves and updates information from the database via the PHP programming language. This visual browsing window does not replace entirely the traditional text-based interface, but provides extra assistance to users. In addition, the size of this applet window can be adjusted to suit each user's preference.

EncCon tree algorithm was used to lay out the structure of the product catalogues. In this visualisation, nodes are used to represent the objects (such as categories, subcategories and auction items), while edges are used to present relationships among the objects or the relations among auction items and categories.

There are several alternative approaches in the design of a navigational structure for online auction sites. The navigational structure can be either *breadth-oriented* or *depth-oriented*. The *breadth-oriented* structure has the advantage of guiding users to their target item with the minimised number of mouse clicks, while *depth-oriented* structure enables the user to browse through more specific sub-category of interesting items effectively. However, the *depth-oriented* navigational structure requires more intermediate levels of retrieval (Hahn 2001). Although the use of appropriate navigational structure purely depends on the nature of applications, our auction prototype system uses *breadth-orientated* structure in its implementation. On the other hand, the navigational structure can be either single-only or multiple hierarchies. The use of multiple hierarchies may increase the chance of locating a target item of interest, but it often confuses the user because of its inconsistency through the site. The navigation structure used in our implementation is a single-only hierarchy.

It is desired that the chosen layout algorithm takes its advantage of geometric space efficiency, speed, and aesthetics. The above features and advantages of our layout technique ensure the capability of handling large or very large scale visualisation with several levels of hierarchical views, i.e. a complex online auction's product catalogue with thousands of auction items. In other words, this could improve the scalability of the traditional interface. The layout also provides an overview of the entire category. This helps users have a better understanding of the overall structure of the product catalogue. Figure 5.3 shows an example of the visual navigational window (that displays all categories, subcategories and auction items) and the main window for browsing online auction's product catalogue.

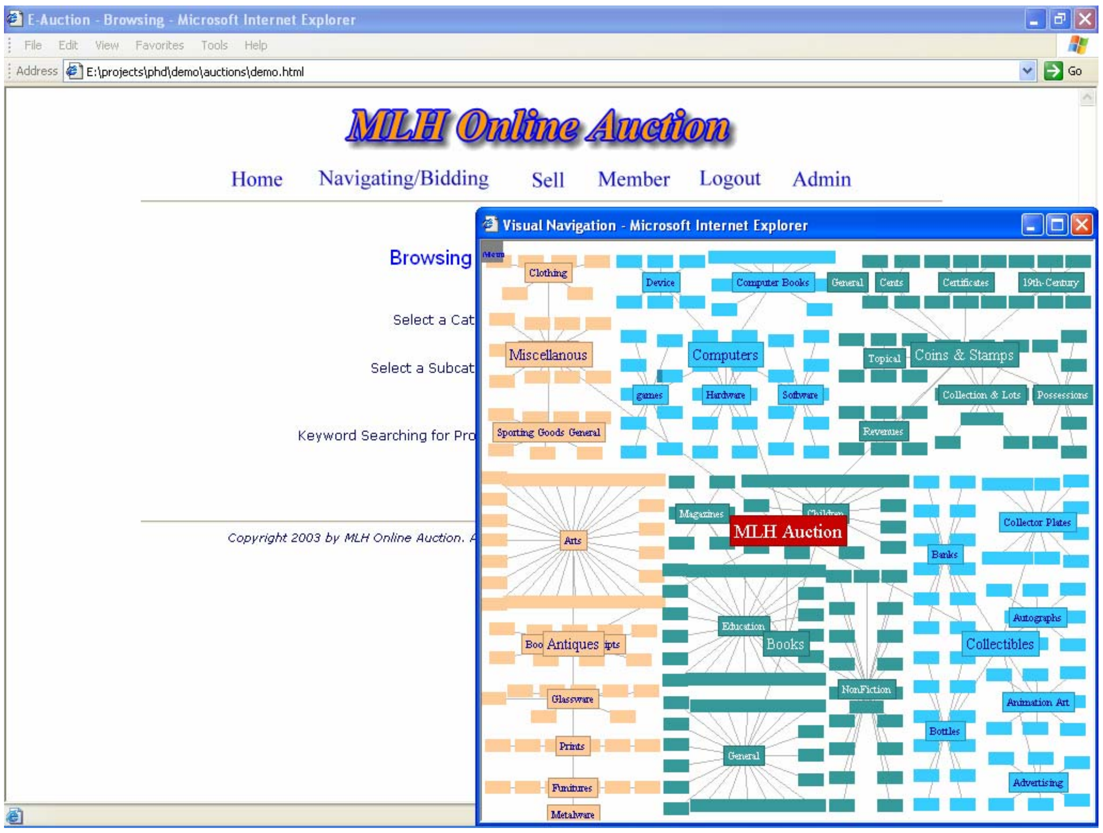


Figure 5.3. An example of the visual navigation window and the main window of the online auction's prototype.

We use our new *layering view* technique for navigating product catalogues. In the visualisation, the layout of the overall context is overlapped with the layout of focused sub-hierarchy. Specifically, a semi-transparent technique is employed to display these dual views in the same geometric area, in which the focused sub-hierarchy is called the detail-view and the context of the hierarchy is called the global-view. This allows users to explore the entire hierarchy quickly by moving around the sub-hierarchies. Each visual interaction is accommodated by an animation in order to preserve the mental-map of the user during the navigation. In more detail, there are two states of the visualisation: *normal* and *context*. Normally, the users' attention is on a particular sub-catalogue from the *detail-view*, and when the *context* state turns on, users' attention moves to the content of the *global-view*. At the *normal* state, the selected sub-hierarchy is displayed with no transparency, while the context is partly transparent and is displayed in brighter colour (see Figure 5.5). At the *context* state, the context is brought from the back to the front and displayed with no transparency, while the detail information is sent from the front to the back and displayed with brighter and partly transparent colour (see Figure 5.6). These two views can be shifted interactively by using a left mouse-click on the background of each layer.

The visualisation uses different colours to present items and subcategories of different categories. The categories and subcategories are also presented with bold boundary to identify auction items within the domain. These displays aim to improve the clarity of the visualisation. The system also provides a mechanism to highlight the new products, ending-today products, and others. This aim to improve the overall display where the users can easily find the special items through the product catalogue (see Figure 5.4). We also provide an interactive menu allowing users to adjust the display to their preferred styles. When the mouse is moving over a node, the sub-hierarchy of the focused node is highlighted to emphasize the selection (see Figure 5.4). In addition, if the focused node is an auctioned item, brief information of this product will be displayed. This property reduces the navigation time since the users can quickly view information of the item from the visual navigation window. In our prototype, the brief information includes current bid price, starting date and closing date (see Figure 5.7). Finally, from the focused item, the bidders can also double-click on a particular product node in order to display all of the information associated with that auction item in the main window (see Figure 5.8).

Figure 5.4 shows a global view of a product catalogue of the prototype system, MLH Online Auction. From this figure, we can quickly identify a new product at the 'Computers-Games' categories. This item is highlighted by being painted with darker colour at their front-end. Figure 5.4 also indicates that the user is focusing on the category 'Computers'. Figure 5.5 shows the next display when the node 'Computers' is selected. You can see from the display of Figure 5.5 that the subcategories and product of the category 'Computers' are enlarged and occupied the entire screen, while the previous context view in Figure 5.4 is reduced and sent to the background with semi-transparency. Figure 5.6 is the display when the global-view active state is selected. One can see that the display is reversed and that the context is brought from the background to the front and displayed with full colours, and the detail-view is sent from the front to the back and displayed with semi-transparent colours.

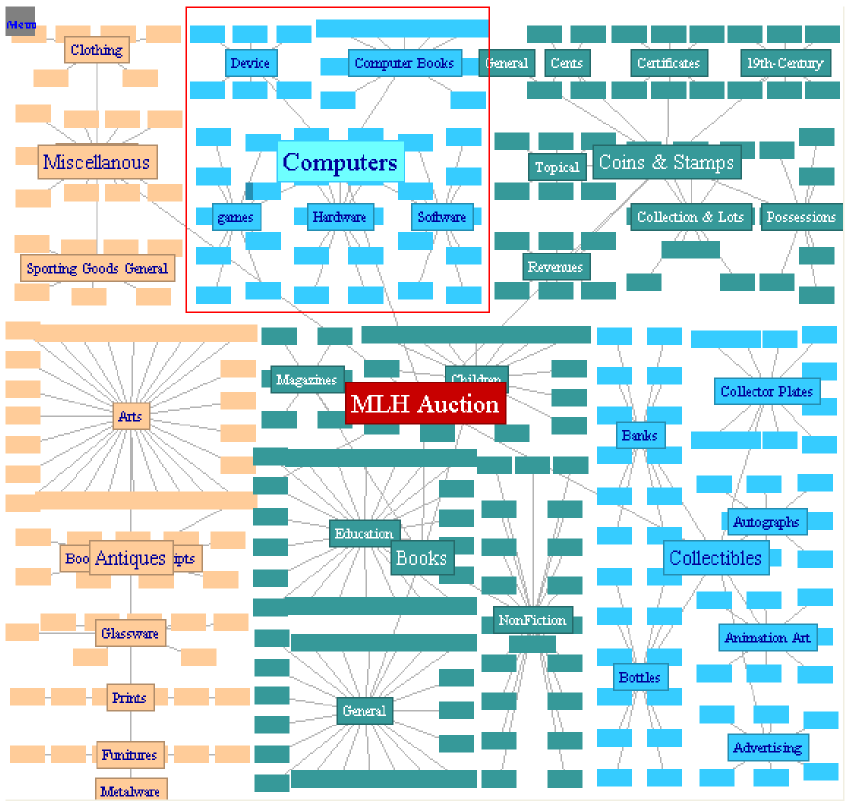


Figure 5.4. The global view of the entire product catalogue.

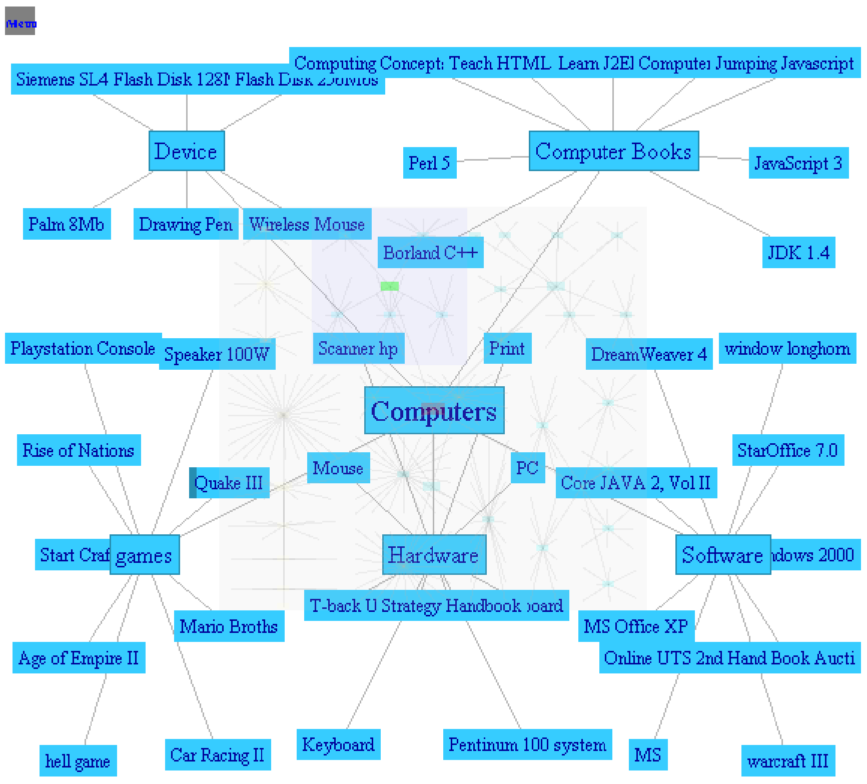


Figure 5.5. The display of all subcategories and auctioned items belonging to the category “Computer”.

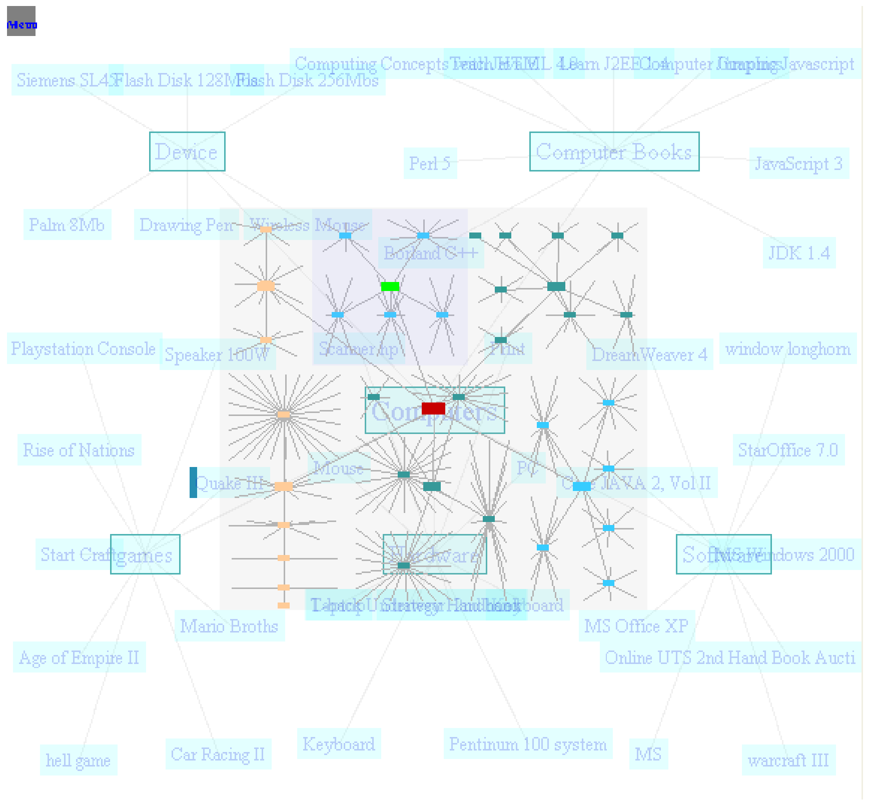
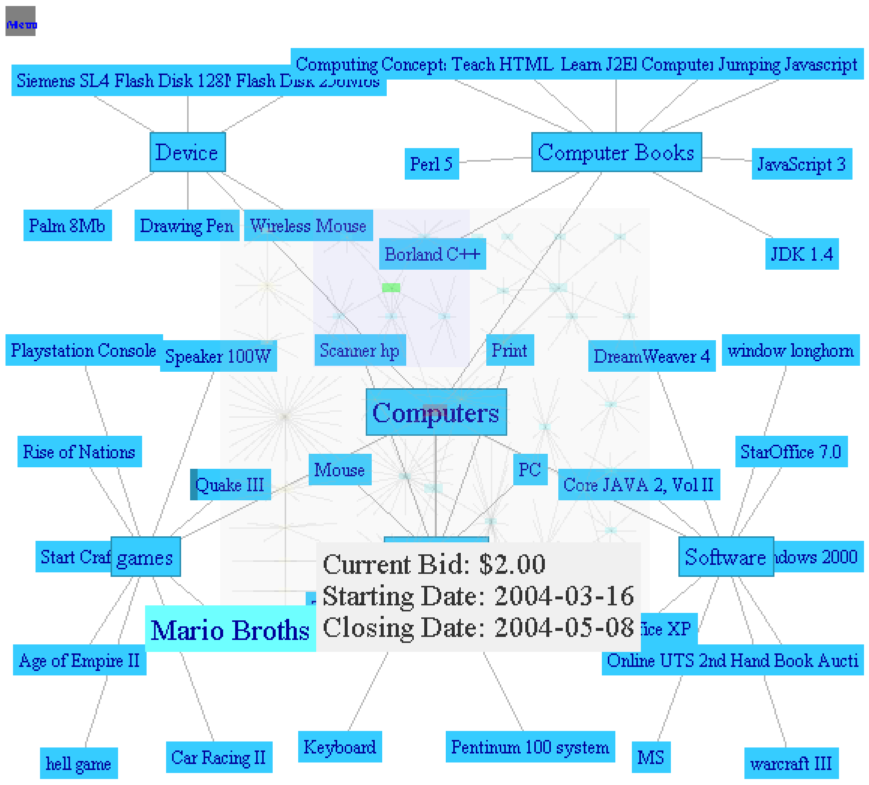
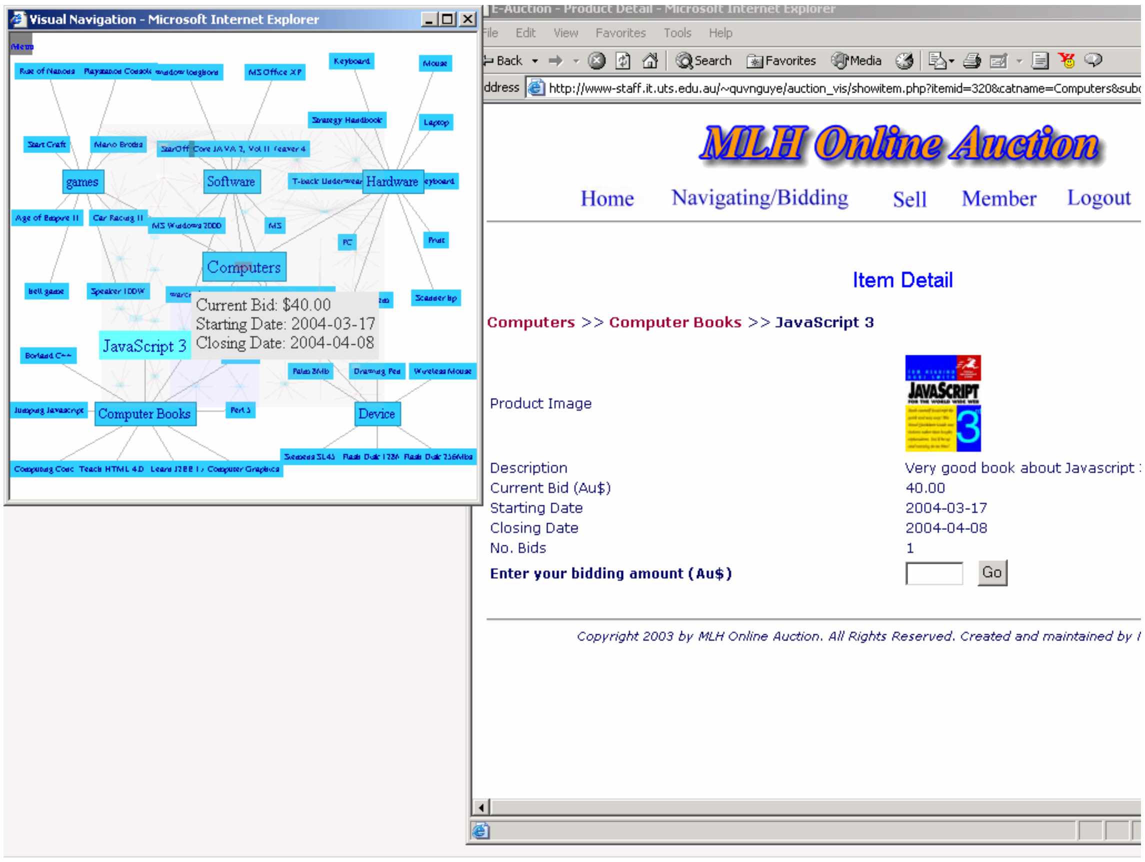


Figure 5.6. The display when the global-view is switched to active state.



Current Bid: $2.00
Starting Date: 2004-03-16
Closing Date: 2004-05-08

Figure 5.7. The display when the mouse is over a product. The system pop-ups a layer to show more detail of the auctioned item.



Current Bid: $40.00
Starting Date: 2004-03-17
Closing Date: 2004-04-08

http://www-staff.it.uts.edu.au/~quvnguye/auction_vis/showitem.php?itemid=320&catname=Computers&subc

Item Detail

Computers >> Computer Books >> JavaScript 3

Product Image	
Description	Very good book about Javascript
Current Bid (Au$)	40.00
Starting Date	2004-03-17
Closing Date	2004-04-08
No. Bids	1
Enter your bidding amount (Au$)	

Copyright 2003 by MLH Online Auction. All Rights Reserved. Created and maintained by

Figure 5.8. An example of the display of both visual navigation and the main window when the bidder double-clicks on the item "JavaScript 3" from categories "Computers-Computer Books".

5.4 Summary

This chapter has presented a new dynamic visual user interface that appears as an additional window and can be used for assisting the online auction process. This visual interface enables users to view and browse the auction catalogue with a large number of the auction items. A new *focus+context* viewing technique called *layering view* is employed to handle the overlapped display of both the *context view* and the *focus view*. This visualisation combines the *EncCon tree* layout and the *layering view* method to provide users with a visual aid for the fast and effective product navigation. Although this application has not been completely finished and commercialised, we believe that this system is very valuable as it can improve the understanding of the product categories during the navigation of auction websites.

Chapter 6 Ongoing Research

This chapter describes two additional techniques, a *three-dimensional EncCon tree* and a *fast navigation technique for the classical hierarchical layouts*, which have been conducted during my Ph.D research. Although these techniques are not the main stream of the study, they are closely related to the research. We now describe these techniques respectively at section 6.1 and section 6.2.

6.1 A Preliminary Three-Dimensional Extension of EncCon Tree

This section describes a preliminary three-dimensional extension of the *EncCon tree* visualisation (see section 2.4). This three-dimensional visualisation includes two parts: layout and navigation. The layout algorithm directly generalises the two-dimensional *EncCon tree* layout algorithm to three-dimensional space in which nodes at the same level of the hierarchy are placed in the same plane. The interactive navigation uses standard three-dimensional viewing techniques which include view transformation, rotation and zoom.

6.1.1 Motivation

With the fast growth of technology, hardware devices for supporting three-dimensional graphics have become more and more powerful. The price of such devices is decreasing rapidly. This makes three-dimensional information visualisation more feasible on a normal personal computer compared to a decade ago.

Chapter 2 has presented two two-dimensional layout algorithms which aim to optimise the display space while retaining the clarity of the display by using a node-link diagram. The success of these layout algorithms motivates me to study the feasibility of extending these algorithms in three-dimensions. The model reuses the above two-dimensional layout algorithms to display the hierarchical data in three-dimensional (3D) space so that nodes at same level are projected onto the same plane. This property aims to provide not only an alternative approach to the visualisation, but also a more realistic look to the three-dimensional model as well as a possible improvement of the clarity in layout.

6.1.2 Layout Algorithm

The layout algorithm of this 3D model is simply a generalisation of *EncCon tree* layout algorithm. This three-dimensional algorithm is described by following processes.

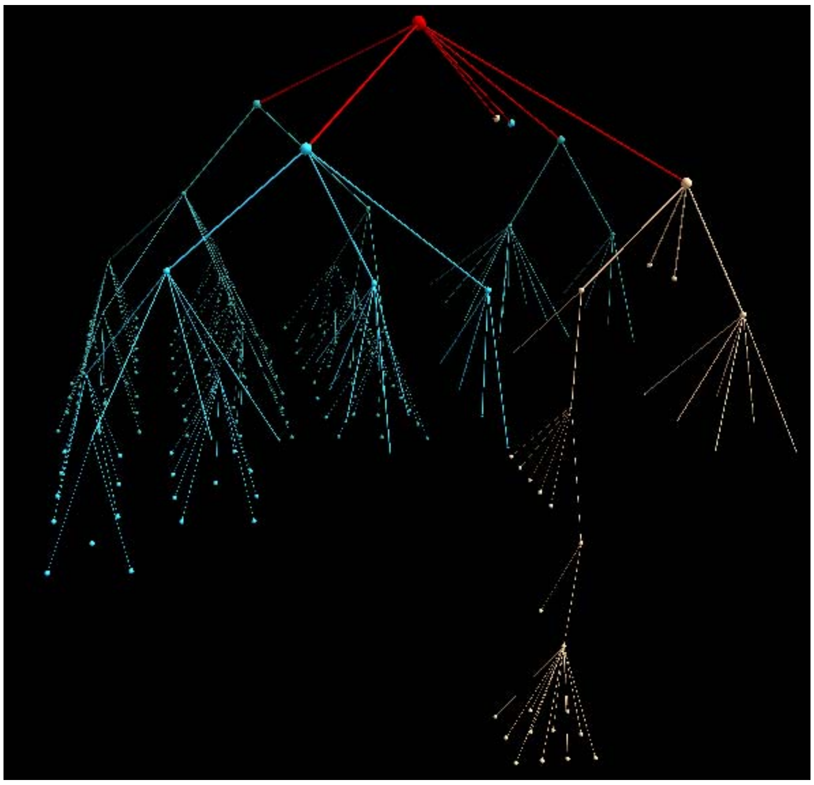
1. First, the x-*, and* y-coordinate value of every node is calculated using the *EncCon tree* layout algorithm (see section 2.4). In this partitioning process, the width and height of the display are normalised as one unit. This step ensures the utilisation of the display space when the layout is projected onto a plane.

2. Next, the z-coordinate value of each node is calculated based on its level that nodes with the same level have the same z-coordinate value. This step is formalised as a projection of the entire hierarchy on the different planes. The number of planes in equal to the number of levels of the hierarchy. These planes are placed along the z-axis. Although the distance between two planes can be adjusted and varied for each level, this prototype only applies the same distance between each two planes. Suppose that the z-coordinate value of the root node is $z = 0$ and the distance between two planes is D, the z-coordinate value of a node at level k is calculated by the formula:

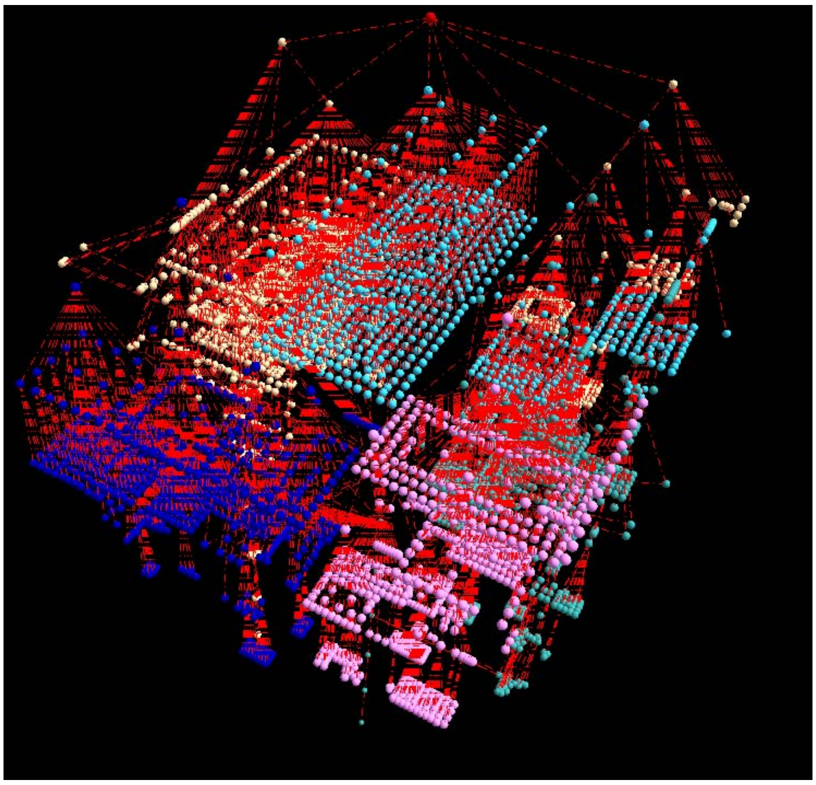
$$z = -kD$$

Equation 6.1

3. Finally, the *(x, y, z)* coordinate value of every node is recalculated to translate from the normalised coordinate system to the display system.

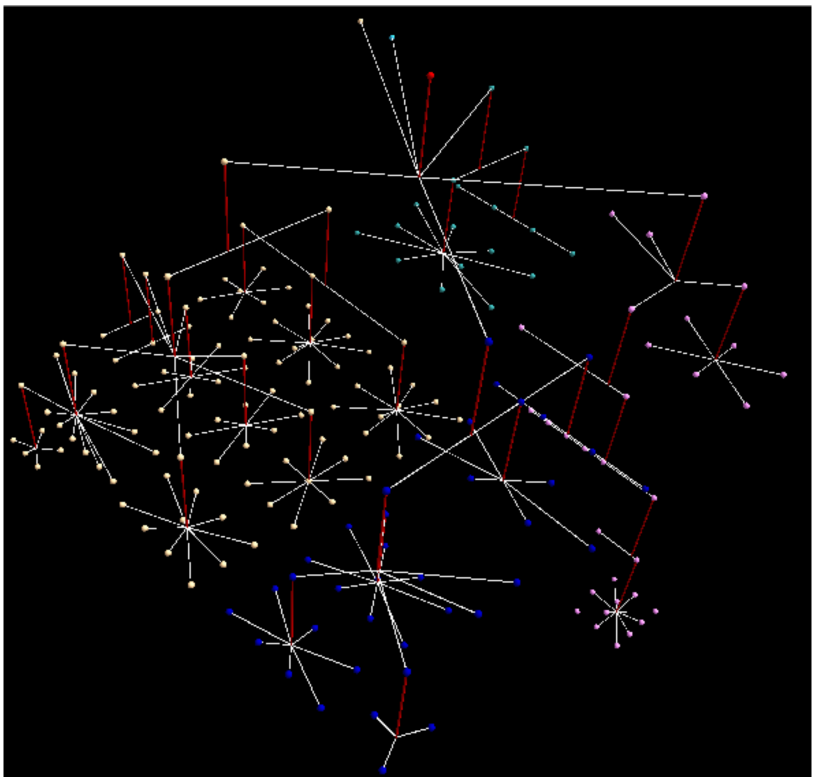


(a)

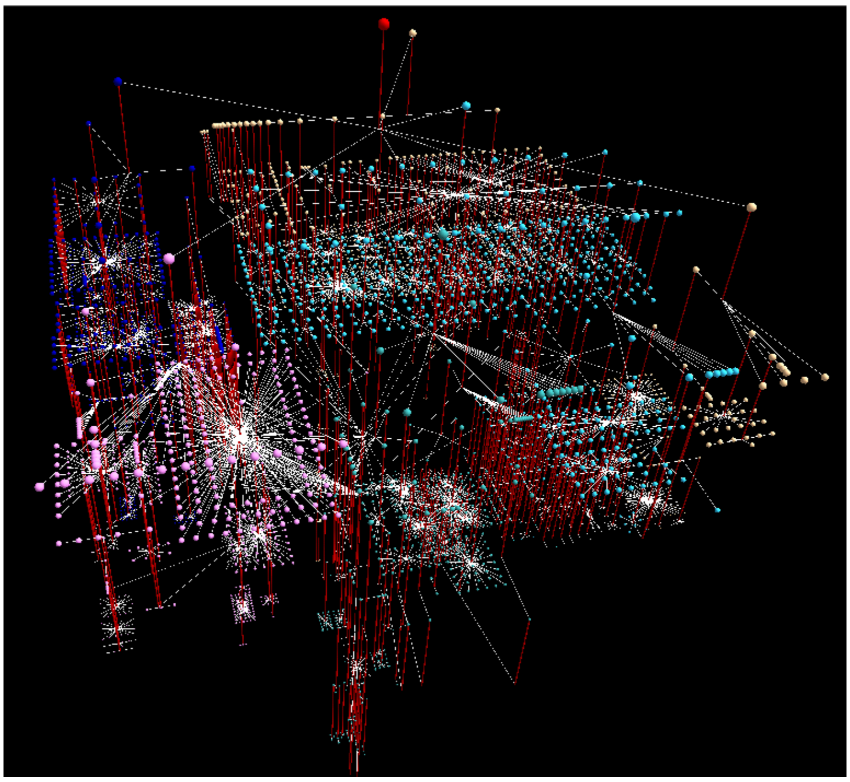


(b)

Figure 6.1. An example of thee-dimensional extension of the *EncCon tree*. That uses the datasets of respectively (a) 170 nodes and (b) 6500 nodes.



(a)



(b)

Figure 6.2. The same visualisation as shown in Figure 6.1 with modified representation of edges.

That uses the same datasets of respectively (a) 170 nodes and (b) 6500 nodes.

Figure 6.1 shows an example of thee-dimensional extension of the *EncCon tree*'s layout that uses the datasets of respectively 170 nodes and 6,500 nodes. Although the three-dimensional visualisation from this algorithm performs reasonable well on several datasets, the overcrowded look of edges could reduce the visualisation quality. This problem normally occurs with a very large datasets that a node might have several child nodes (see Figure 6.1b). To overcome this limitation, we modify the representation of edges by replacing *straight-line links* with *bended links*. Technically, an edge is not drawn directly from a parent node to a child node through a straight-line link, but a bended link which includes two segments. The first segment is drawn from this node to the centre of the rectangular local area of its child nodes. And the second segment is drawn from this centre point to a child node. Figure 6.2 shows an example of the visualisation from this modification that uses the same datasets as Figure 6.1.

6.1.3 Summary

This section has presented an ongoing three-dimensional extension of the *EncCon tree* visualisation. The layout algorithm used for this visualisation extends the original two-dimensional *EncCon tree* to a three-dimensional space. This property aims to provide not only an alternative approach of the visualisation, but also a more realistic look of the three-dimensional model as well as a possible improvement of the clarity in layout. We use two different edge representations, the straight-line and the bended-line, to implement this three-dimensional *EncCon tree*. This three-dimensional system is just at the initial state and there is still need for improvement. However, this initial work shows a great potential in developing alternative *EncCon tree* visualisation.

6.2 A Fast Focus+Context Viewing Technique for the Navigation of Classical Hierarchical Layout

This section presents a fast *focus+context* viewing technique for the navigation of classical hierarchical layouts (see an example at Figure 1.3). In classical hierarchical layouts, the density of display information expands only in one dimension. Based on this property, we only consider one dimension of distortion of views for the navigation of hierarchical structures. This section first discusses the features, advantages and limitations of layout and navigation of current classical hierarchical layouts. It then describes new a *focus+context* technique for the navigation and exploration of the classical hierarchical layouts.

6.2.1 Classical Hierarchical Layout

The classical hierarchical layout is based on the algorithm developed by (Reingold & Tilford 1981) which is also called the *RT* algorithm. In this algorithm, children nodes are positioned below their common ancestor (see Figure 1.3). In short, the *RT* algorithm calculates the relative position of sub-trees independently and then it joins them in a larger tree by placing these sub-trees as close as possible. The detail of this algorithm is described at the section 6.2.2.

Currently, several newly two-dimensional layout techniques have been proposed and developed such as *radial view* (Eades 1992), *balloon view* (Melancon & Herman 1998), *hyperbolic browser* (Lamping & Rao 1996; Munzner 1997), *disk tree* (Chi et al. 1998), *NicheWorks* (Wills 1999), *rings* (Teoh & Ma 2002), and others. These techniques are often superior in term of space-efficiency, navigation, and large-scale visualisation. However, these new techniques cannot match the *RT* algorithm in terms of simplicity, predictability and aesthetics of trees visualisation. As a result, the *RT* algorithm is widely used in many applications where datasets are reasonably small.

The hierarchical layout, however, still has limitations that its layout tends to be too wide to be displayed within a screen size. As a result, the classical hierarchical layout is not very applicable to large datasets. There are a few variations to this layout technique which was described by (Kennedy 1996), (Bruggemann-Klein & Wood 1988), and (Herman, Delest & Melancon 1998), etc. In addition, there are few good available navigation techniques for viewing large classical hierarchical layouts. Typical viewing techniques available for navigating classical hierarchical layouts are *zooming* (Herman, Delest & Melancon 1998) and *fisheye-view* (Furnas 1986). When a focused area is zoomed, the overall context is lost. This might lead to a broken mental map during the navigation in the zooming techniques. On the other hand, *fisheye-view* is a *focus+context* technique where the context is kept during the navigation. However, this technique causes the distortion of views which might reduce the quality of the visualisation of both context and detail information. Furthermore, most classical *focus+context* methods use two-dimensional distortion technique for viewing hierarchies. This technique costs considerable a large amount of the computation time for view transformations during the navigation, especially for viewing large hierarchies. This problem sometimes slows down the navigation activities.

(Plaisant, Grosjean & Bederson 2002) described an excellent navigation technique for navigating hierarchical layouts. This technique provides a browsing mechanism allowing a user to interactively move the selected sub-hierarchy into the centre of the display area. The algorithm also enlarges and displays more detail the focus sub-tree, and it reduces and filters the others. However, this technique does not apply the *RT* algorithm in its implementation which is the main concern of my research.

This section describes a fast and simple *focus+context* technique for the classical hierarchical tree visualisation. This technique applies *Reingold & Tilford* algorithm in its layout and it attempts to retain the shape of the layout during the navigation. This addresses the problem of preserving a mental map. Specifically, the hierarchical layout is presented in vertically top-down manner where nodes in the equal-level are placed at the same horizontal line. The navigation operates horizontally and focus regions are dragged into the middle. This reduces the computation time for changing views as we only consider one-dimensional distortion of views for the browsing of hierarchies. A mechanism is also provided to solve the problem of labelling, especially the problem of displaying a very long node label.

6.2.2 Technical Detail

Our viewing technique can be applied to any layouts where only one-dimensional distortion is considerably important. In the scope of this paper, we only apply our viewing technique to rooted trees where their layouts are based on *Reingold and Tilford* algorithm (Reingold & Tilford 1981). This layout algorithm was chosen to demonstrate our viewing technique because it is simple, fast, well known and produces a nice layout overall. The system includes two major sections: layout and interactive navigation. The layout section is responsible for constructing the tree layout while the interactive navigation section is used for the viewing and navigation of these hierarchies.

6.2.2.1 The Layout

There are two separate steps in this section. The system first constructs a tree layout based on the *RT* algorithm. A transformation is then applied to every node in order to achieve the final drawing.

The *RT* algorithm takes a modular approach to calculate positions of nodes. The relative positions of the nodes in a sub-tree are calculated independently from the rest of the tree. Conceptually, the algorithm recursively traverses through the tree and place the nodes from leaves upward to the root of the tree. The process uses bottom-up direction in our system. Each sub-tree is considered as a local unit. From left to right direction, the siblings of a node in a sub-tree are placed at a proper minimal distance from one another. Then, all nodes of each sub-tree are shifted to the right in order to avoid intersection between sub-trees (see Figure 6.3).

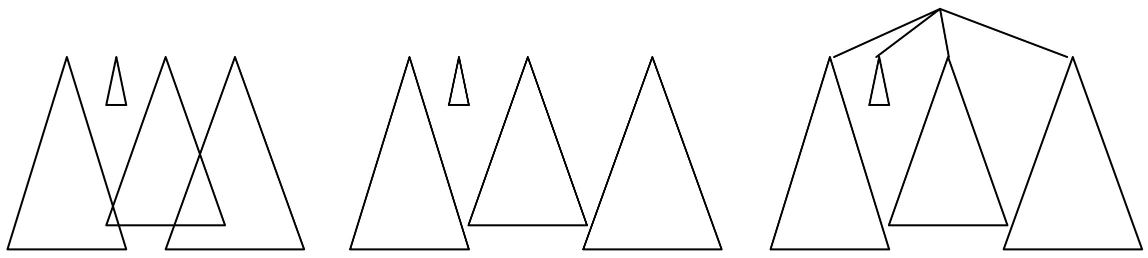


Figure 6.3. The Reingold and Tilford algorithm.

The shifting of all nodes in the sub-tree is time consuming. To overcome this problem, Reingold and Tilford used the concept of a preliminary x-coordinate and a modifier field for each node. The tree placement is recursively calculated in two separate traversals, namely post-order and pre-order traversals. In the first tree traversal, the intermediate x-coordinate and the modifier value are set. The modifier value is used to indicate the amount to be added to the intermediate x-coordinate of all nodes of the corresponding sub-tree. Then, the pre-order traversal is applied for calculating the final x-coordinate values of each node by accumulating the modifier values in the process. This traversal also calculates the y-coordinate values based on the level of each node in the hierarchy. The detail of this algorithm is described in the conference paper titled ‘Tidier drawing of trees’ (Reingold & Tilford 1981). Figure 6.4 shows an example of the display when the *RT* algorithm is applied to visualise a hierarchical data structure.

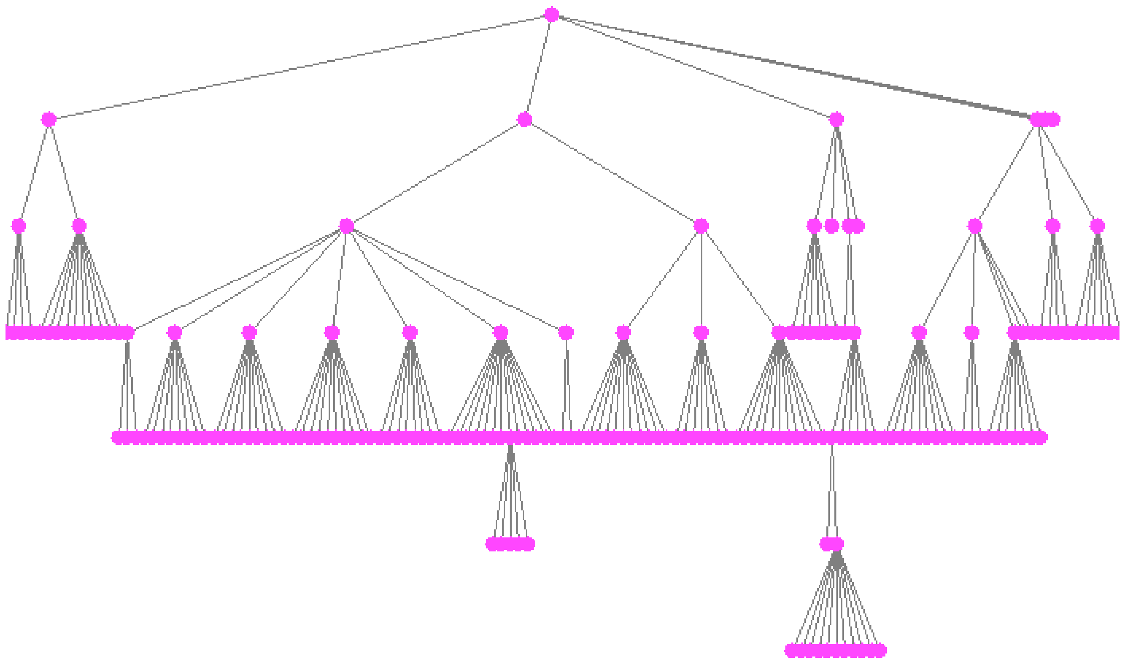


Figure 6.4. An example of displaying a tree layout using *RT* algorithm.

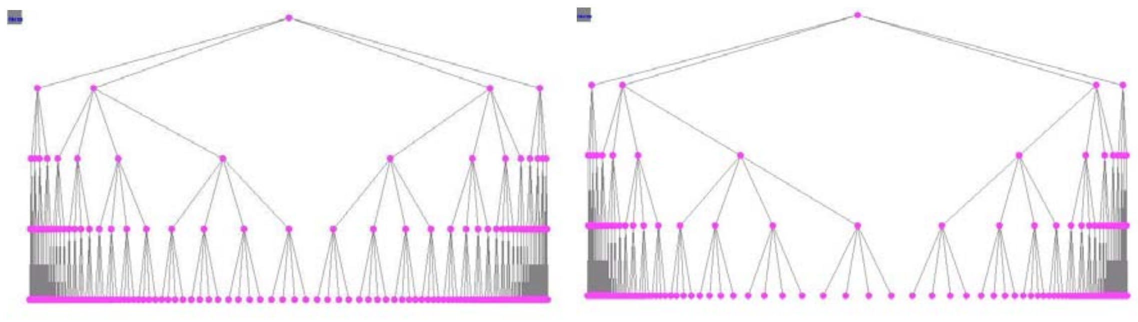
The final displacement of the hierarchy is then calculated based on values from the above *RT* algorithm. For each node of the hierarchy, we apply a transformation function in order to get the final coordinate value. In our system, the point (0, 0) is located at the top-left of the display window. For each node *N*, suppose that (x_{rt}, y_{rt}) is the coordinate of node *N* after using *RT* algorithm. The final coordinate (x, y) is calculated by the formula in Equation 6.2:

$$\begin{cases} x = \dfrac{2C\tan^{-1}(M(x_{rt} - C))}{\pi} + C \\ y = y_{rt} \end{cases}$$

Equation 6.2

where *C* is a constant and is equal to half the width of the display window. In other words, *C* is the offset value of two coordinate systems (*transformation* and *normal*). In our *transformation coordinate*, the value x = 0 is defined at the centre of the window in horizontal direction. *M* is also a constant and it defines the magnification of the distortion. The magnification is proportional to the value of *M*. This magnification value can be adjusted during the navigation in order to archive a suitable view. Figure

6.5a and Figure 6.5b are two examples of the final display, where respectively *M = 0.02* and *M = 0.04*.



(a) (b)

Figure 6.5. An example of the layout where (a) M = 0.02 and (b) M = 0.04.

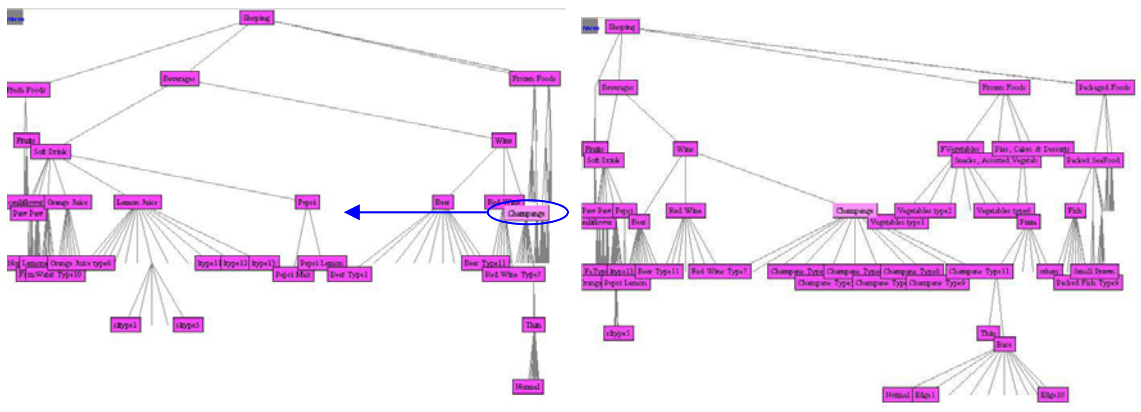
6.2.2.2 Interactive Navigation

In our system, the navigation operates horizontally by dragging information to the focus area, i.e. the centre area. Suppose that the mouse drags in the horizontal direction a distance *K*. For each node *N* in the tree, its x-coordinate value is recalculated in Equation 6.3.

$$x = \frac{2C\tan^{-1}(x_{old} + MK))}{\pi} + C$$

Equation 6.3

where *C* and *M* are constants that are defined at Equation 6.2, and x_{old} is the original x-coordinate value of node *N*. The y-coordinate value is unchanged during the navigation. Figure 6.6a is an example of the original display of a large dataset. Figure 6.6b shows the display when focused node, labelled 'Champagne', is dragged to the centre area.



(a) (b)

Figure 6.6. An example of the navigation and interaction.

Figure (a) shows the original display, and figure (b) shows the display when a node labelled 'Champagne' in (a) is dragged to the centre.

6.2.2.3 Display Property

In our system, we only display the detail of nodes that are not very close to their siblings. Similarly to the classical hierarchical layout technique, we also have problems with labelling when the label of a node is very long. To overcome this problem, a node label is drawn just below the normal position if the label is too long and it clashes with others (see Figure 6.6). This property aims to improve the clarity of layout overall.

6.2.2.4 Complexity

The above layout algorithm includes two separate steps: the first step uses *Reingold* and *Tilford* algorithm and a transformation function. The run time complexity of *Reingold* and *Tilford* algorithm is *O(N)* where *N* is the number of nodes (Reingold & Tilford 1981). The transformation function presented in Equation 6.2 is linearly applied to every node of the hierarchy which also costs *O(N)*. As a result, our layout complexity is *O(N)* or linear. The navigation and interaction also applies a transformation function to every node as Equation 6.3. This means that the computational cost of each navigation is linear or *O(N)*. In conclusion, the complexity of both layout and navigation is linear or *O(N)*.

6.2.3 Summary

This section has presented a fast *focus+context* technique for the viewing and navigation of classical hierarchical layouts in which only one dimensional distortion of view is considerably important. Our technique employs a simple and fast one-dimensional distortion of views to help users browse and retrieve the hierarchical information without losing the context view. The *Reingold & Tilford* layout algorithm was chosen for the implementation of our technique because this algorithm is simple, fast, predictable and it produces a nice-layout overall. Although this technique still needs to be improved, we believe that it is a good method for navigating hierarchical information with the classical hierarchical layout.

Chapter 7 Conclusions and Future Work

The preceding chapters have presented, discussed and illustrated all of the technical elements of this research. The current chapter concludes the thesis by providing a summary of the main points and the major contributions of this thesis in information visualisation. Finally, this chapter outlines the areas for future work and the sketch plan of two usability studies.

7.1 Conclusions

Most existing visualisation systems do not consider the issue of increasing the utilisation of display space for visualising large graphs and/or hierarchies. Hierarchical layouts generated by the existing visualisation systems usually contain a large portion of the screen pixels that are wasted as background (see detail discussion at section 1.3). Consequently, we are unable to fit large amount of contextual information (the *global view*) with thousands of items onto a computer screen. The traditional *virtual page* solution used by many existing systems, however, breaks the *global view* of the information into several windows. This makes it difficult for users to understand the scope of information spaces or to find where they are and where they should go to access information they need. Furthermore, this *virtual page* approach requires a huge memory to store and display the large virtual screen. An effective strategy to improve this situation is to increase the utilisation of display space to allow more nodes/edges to be fitted into the screen.

This thesis has presented a new space-efficient model and associated algorithms to handle the visualisation of large hierarchies in a two-dimensional space. It is the first attempt to use a space-efficient approach to address the problem of visualising large hierarchies which differs from other approaches. Our new model inherits the advantage of efficient space utilisation from the *enclosure* approach while it tries to retain the clarity of hierarchical representation by using node-link diagrams. This approach does not require high hardware support, such as memories, screen resolution and computational power, which makes it possible to be run on most ordinary personal computers. We have shown that this approach could be one of the cost-effective ways to solve the problem in comparison with other approaches.

We next summarise the main points of our layout algorithms and navigation methods, as well as the major contributions of the thesis.

7.1.1 Space-Efficient Visualisation

Traditional hierarchical visualisation algorithms are more concerned with the readability of drawings. This can be measured by a set of aesthetics, such as the minimisation of edge crossings, total edge length and the variance of the lengths of the edges. They usually do not consider the efficient utilisation of the geometrical plane for drawings. However, when the user wishes to have the entire layout of a large graph with (tens of) thousands of nodes/links displayed within a single screen, the efficient utilisation of the geometrical plane becomes a crucial issue and more important than aesthetics in the production of large layouts. We first need to consider how to fit a large number of nodes/links on a screen before we should consider the aesthetics needed to enhance the readability of layouts. In this research, we have investigated new hierarchical drawing algorithms that aim to produce high quality layouts of large hierarchies, which could satisfy both the aesthetic rules of graph drawing and utilisation of the geometric plane thus allowing more nodes/links to be displayed on the screen.

This thesis has presented two layout algorithms for visualising large hierarchical information. They are *space-optimised tree* and *EncCon tree* that support the space-efficient principle. *Space-optimised tree* (or *SO tree*) is an effective partitioning algorithm for laying out large hierarchical information. This algorithm optimises the space to display information at a limited screen resolution. It partitions the geometrical layout as polygons in which each polygon is formed by the intersection of a wedge and the given polygonal local region. *Space-optimised tree* is quite capable of visualising the entire tree structure of large datasets at the available screen resolution (see examples from Figure 2.7 to Figure 2.10).

EncCon tree is another enclosure partitioning algorithm that uses a rectangular space-filling method for recursively positioning of trees in the display space. The space-filling method used in the *EncCon tree* is similar to *Squarified Tree-Maps* (Bruls, Huizing & van Wijk 2000). In comparison with the above *SO tree*, the rectangular space-filling method is generally easier for viewers to perceive hierarchical relationships. Furthermore, the rectangular space-filling algorithm used in the *EncCon tree* is relatively simple and requires less computational time than the polygonal space-filling algorithm.

In Chapter 2, we have also conducted an experimental evaluation of *EncCon tree*'s partitioning algorithm in comparison with several *tree-maps* algorithms. The comparison result shows that only the 'square-like' partitioning algorithms, including the *EncCon tree* and the *squarified tree-maps*, are better for our *enclosure+connection* visualisation. Further analysis using graph drawing aesthetic rules at section 2.4.7 also indicates that the *EncCon tree* algorithm often performs better than the *squarified tree-maps* algorithm, especially for large datasets (see Figure 2.28).

7.1.2 Navigation Methods

Navigation of large hierarchies is the second key issue raised in the design of interactive visualisation. Even if the advances in graph drawing research have enabled the efficient geometrical positioning of trees and their displays, a visualisation of large trees would still be useless in terms of assisting users to retrieve particular data items, unless they provides efficient navigation mechanisms in collaboration with the layout algorithms. As a result, navigation and interaction techniques are generally implemented in information visualisation system in conjunction with the layouts.

In Chapter 3, we have described the technical detail of two new interactive navigation methods, called *hybrid view*, and *layering view*. These two techniques use *focus+context* approach by which both the global and the detail information are displayed concurrently. Although these interactive navigation methods are independent to the layout algorithms, the prototypes we used to demonstrate these methods are based on the *space-optimised tree* and *EncCon tree* layout algorithms.

The *hybrid view* technique includes a *browsing* algorithm and a simplified fisheye-like distortion algorithm. The *browsing* section is responsible for bringing a sub-hierarchy to the focus area, while the *distortion* section enlarges the information at the focus area. This technique also applies *semantic zooming* to reduce the amount of visual information when the display is overcrowded. The low computational-cost property of *hybrid view* speeds up the navigation process allowing a very large dataset to be quickly visited on a normal personal computer.

Layering view uses a new *zooming+layering* concept (see section 3.2.1) to achieve the *focus+context* viewing of large hierarchies; rather than the traditional *enlarge+embedded* approach which is used by most of the existing *focus+context* techniques. As a result, this technique is generally better in term of utilisation space for displaying information. Technically, it employs a semi-transparent graphical technique to achieve the concurrent display of two separate layers of information on the same physical screen. This alternative approach enlarges the display areas for displaying context view and detailed view and allows more information to be presented in both views. The interactive navigation is achieved by semantic zooming, updating views and swapping views between layers. All these transactions are accommodated by animation to preserve the user's mental map of the views.

7.1.3 Case Studies

In Chapter 4, we have presented the first application of our layout and navigation techniques in the domain of collaborative workspaces. This visualisation component appears as an additional window to the main text-based user interface, and can be used for viewing, browsing, analysing, and manipulating *LiveNet* collaborative workspaces. This visual interface consists of five main techniques: 1) a modified *EncCon tree* layout algorithm, 2) a semi-transparent viewing technique, 3) a clustered viewing technique, 4) an animated *focus+context* interactive navigation technique, and 5) a set of rich graphical techniques, including icons, pop-up windows, shapes, sizes, colours, brightness, to visualise the attributed properties of the collaborative workspaces. The system visualises multiple relationships among collaborative objects of shared workspaces. These types of relationships include *categorising*, *accessing*, *participating*, *sharing* and *occupying*. These relationships are concurrently displayed at a single display space using the above techniques. The visualisation also employs several graphical techniques to visualise the attributed properties of all collaborative objects and relations in the workspaces. Therefore, this visual interface provides the users with a better assistance for visual learning, manipulation and navigation in collaborative workspaces. In addition, this chapter also describes an approach for navigating, highlighting, as well as viewing the associative information of workspace elements.

In Chapter 5, we have described the second application of our new visualisation in the domain of e-business. This system is a prototype of a visual interface for viewing and navigating product catalogues of large-scale online auction websites. This system also employs *EncCon tree* layout algorithm to display the entire as well as a portion of product hierarchy. This visual interface provides not only a *focus+context* viewing technique to allow a user to browse the entire the product catalogue, but also a mechanism for quickly analysing and recognising special auctioned items. Consequently, this system has shown its potential for assisting the user in online auctions.

7.1.4 Ongoing Research

In Chapter 6, we have presented two additional studies which have been recently conducted. These include a preliminary three-dimensional extension of *EncCon tree* layout algorithm and a fast *focus+context* viewing technique for the navigation of classical hierarchical layout. Although these techniques are not the main focus of this research, they are natural extensions of the core research and showing the potential value as alternative research directions for my future research.

7.2 Future Work

The preceding sections have summarised all the main points and the major contributions of this thesis in information visualisation. This section outlines the possible directions of future work from this research.

As part of this research, we have made a preliminary three-dimensional extension of *EncCon tree* layout algorithm. Although this initial prototype shows its potential, the work we have done is still in the early stage. In the future we will improve our initial prototype and make a formal study of three-dimensional hierarchical visualisation. We will also conduct a usability study to evaluate this three-dimensional technique.

In the thesis, we have demonstrated that our two-dimensional *EncCon tree* is an effective approach for laying out large tree structures. However, there are still some problems need to be improved in the near future, such as the problem of overlapping among nodes.

The author will investigate optimised viewing techniques and layout algorithms to minimise the effect of our *layering view* in human cognition processes. A usability study will also be carried out to evaluate the effectiveness of transparent layering for visualising graph or hierarchical data. Although the layering navigation technique is independent of the layout, it works better with optimised layouts of the node-link diagram. We believe that by adopting optimised layout algorithms, the overlaps among nodes between the *context view* and the *detail view* will become insignificant.

Although an experimental evaluation has been conducted to evaluate our partitioning and layout algorithms through the comparison of several *tree-maps* algorithms, a usability test for our new navigation methods that enable users to explore large scale visualisation efficiently will be carried out in the near future. The test will try to compare several human perception aspects of our method with other navigation methods, such as the *fisheye view*. Some initial sketch of the usability studies is further described.

7.2.1 Usability Study – Topology of Tree

7.2.1.1 Type of Layouts

The experiment evaluates our *space-optimised tree* (Nguyen & Huang 2003), and *EncCon tree* (Nguyen & Huang 2005) layouts against the other typical tree visualisation techniques. Two popular layout techniques are selected in our experiment. They are the *classical hierarchical layout* (Reingold & Tilford 1981) and *tree-maps* (Johnson & Shneiderman 1991) which represent connection approach and enclosure approach respectively.

7.2.1.2 Criteria for Evaluating Layouts - Topology of Tree Visualization

- *Ease of interpretation:* this indicates how well a user understands the parent-child and sibling relationships of any tree. The user should be able to recognise the property of the relational hierarchies, such as identifying the largest sub-tree, identifying the number of levels of a tree, by viewing the entire display.
- *Comparison of node size:* this indicates how well a user recognises a node's property.

- *User preference:* this indicates the user preference of above tree layout interfaces.

7.2.1.3 Design

The experimental design is a variation of a 4 (views) x 5 (tasks), repeated-measures the interfaces with different tree datasets as a random factor. This experiment is applied for medium to large tree hierarchies.

7.2.1.4 Participants

- *Non-expert participants:* selected from computer science undergraduate or postgraduate students who are not or little familiar with information visualisation, especially in hierarchical information visualisation including *classical hierarchical view*, *tree-maps*, *space-optimized tree*, and *EncCon tree*.
- *Expert participants:* selected from research students who are familiar with information visualisation, especially hierarchical information visualisation.

7.2.1.5 Tasks

We design 5 tasks to test the ability of the view to communicate the tree topology.

4. *Binary or n-ary:* participants have to design if the tree is binary or n-ary (more than 2 children at a node). All the trees in this task are not uniform.
5. *Balanced or unbalanced:* participants decide if a tree is balanced or unbalanced.
6. *Deepest common ancestor:* participants decide which node is the deepest common ancestor of two highlighted nodes.
7. *Number of levels:* participants count the number of levels in the tree.
8. *Three largest sub-trees:* participants decide three largest sub-trees in descendant order from a given tree.

7.2.1.6 Procedure

- All the figures from this test have size of 700x700 pixels.

- Screen size is a standard 17 inches monitor with 1024x768 pixels resolution.
- Eight samples datasets are selected in the experiment.
- Two samples are randomly selected on each task.

7.2.1.7 Measurement

- *Response time:* the time to answer questions.
- *Accuracy:* correctness.

7.2.1.8 Questionnaires

- Q.1. *is the tree binary or n-ary?* A binary tree is a tree which all of the parents have almost 2 children. An n-ary tree is a tree which most of the parents have more than 2 children.
- Q.2. *is the tree balanced or unbalanced?* A balanced tree is a tree which has leaves on the same level or two consecutive levels.
- Q.3. *what is the deepest common ancestor of the two highlighted nodes?* The deepest common ancestor of two nodes is the node which is the closest direct ancestor of these two nodes.
- Q.4. *how many levels does the tree have?* The number of levels of a tree is the level of the deepest leaf. The root level is 1.
- Q.5. *select three largest sub-trees in order 1st, 2nd, and 3rd?* A sub-tree of a tree is the tree whose root is the node having level 2 or level 3 (if there are only one node at level 2). The largest sub-tree is a sub-tree which has most number of descendants. Only two sub-trees are listed if the tree is a binary tree.

7.2.2 Usability Study – Navigation and Topology of Tree

7.2.2.1 Type of Tree Visualisation Systems

This experiment evaluates the *EncCon tree* (Nguyen & Huang 2005) against three popular tree-browsing techniques including *hyperbolic browser* (Lamping & Rao 1996), *tree-maps* (Johnson & Shneiderman 1991), and *space-tree* (Plaisant, Grosjean & Bederson 2002).

7.2.2.2 Criteria for Evaluating Tree Visualisation Systems

- *Navigations:* this indicates how fast and effective a user navigates through a very large hierarchical tree.
- *Topology:* this indicates how well a user understands the topology the tree beside the navigation.
- *User preference:* this indicates the user preference of above systems.

7.2.2.3 Design

The experiment uses four data sets (file systems) which are carefully selected to ensure they are similar in terms of number of levels traversed and semantic complexity of the data explored. We randomly selected one data set for each experimental system.

7.2.2.4 Participants

- *Non-expert participants:* selected from computer science undergraduate or postgraduate students who are not or little familiar with information visualisation, especially in hierarchical information visualisation including *hyperbolic browser*, *tree-maps*, *spacetree*, and *EncCon tree*.
- *Expert participants:* selected from research students who are familiar with information visualisation, especially hierarchical information visualisation.

7.2.2.5 Tasks

We design 3 tasks to test the ability of navigating through a hierarchy and 2 tasks of communicating the tree topology. These tasks are further described.

9. First-time node finding: this task is to find 3 nodes which have never seen before.
10. Returning to previously visited nodes: this task is to find the first visited node from the first task.

11. Listing all the ancestors of a node: this task involves when the user is finished task 2.
12. Finding nodes with property: this task is to find three nodes which have more than fifteen direct children.
13. Topology of tree: this task is to find the three sub-hierarchies with most number of nodes.

7.2.2.6 Procedure

- All the programs running from this test have size of 700x700 pixels.
- Screen size is a standard 17 inches monitor with 1024x768 pixels resolution.
- Four computers are used and each of them is applied for one visualisation technique using one random data set from 4 samples. No same data set is used in any two techniques to ensure the minimisation of learning effect.
- A stop watch is used to count the reaction time.
- Four different datasets are used with similar number of levels traversed and semantic complexity.

7.2.2.7 Measurement

- *Response time:* the time to answer questions.
- *Accuracy:* correctness.

7.2.2.8 Questionnaires

- Q.1. *first-time node finding:* this task is to finding three nodes which have never seen before.
- Q.2. *returning to previously visited nodes:* this task is to finding the first visited node from task 1.
- Q.3. *listing all the direct ancestors of a node:* this question involves when the user is finished task 2.

- Q.4. *finding nodes with property:* this task is to finding three nodes which have more than twenty direct children (this apply to the entire data set).
- Q.5. *topology of tree:* finding the three sub-hierarchies with most number of nodes.

Author’s Publications for the Ph.D

Journal Papers

- Nguyen, QV & Huang, ML 2005, 'EncCon: An Approach to Constructing Interactive Visualization of Large Hierarchical Data', *Information Visualization Journal (Palgrave Macmillan)*, vol. 4, no. 1, pp. 1-21.
- Nguyen, QV & Huang, ML 2003, 'Space-Optimized Tree: a Connection+Enclosure Approach for the Visualization of Large Hierarchies', *Information Visualization Journal (Palgrave Macmillan)*, vol. 2, no. 1, pp. 3-15.

Refereed Conference Papers

- Nguyen, QV & Huang, ML 2005, 'DualView: A Focus+Context Technique for Navigating Graphs', *International Conference Computer Graphics, Imaging and Vision (CGIV’05)*, IEEE, Beijing, China, pp. 162-166.
- Huang, ML, Nguyen, QV & Hintz, T 2005, 'Attributed Graph Visualization of Collaborative Workspaces', *International Conference Computer Graphics, Imaging and Vision (CGIV’05)*, IEEE, Beijing, China, pp. 155-161.
- Nguyen, QV & Huang, ML 2004, 'A Combined Visualization of Multiple Relational Structures in Shared Collaborative Workspaces', *2004 International Workshop on Multimedia and Web Design - in Conjunction with The 6th International Symposium on Multimedia Software Engineering*, IEEE Computer Society, Miami, Florida, USA, pp. 388-395.
- Nguyen, QV & Huang, ML 2004, 'An ENCCON Visual Browser for Large-Scale Online Auctions', *International Conference on Internet Programming (IC'04)*, vol. 2, CSREA, Las Vegas, USA, pp. 558-564.
- Nguyen, QV & Huang, ML 2004, 'A Focus+Context Visualization Technique Using Semi-Transparency', *The Fourth International Conference on Computer*

and Information Technology (CIT 2004), IEEE Computer Society, Wuhan, China, pp. 101-108.

- Nguyen, QV, Huang, ML & Hawryszkiewycz, I 2004, 'A New Visualization Approach for Supporting Knowledge Management and Collaboration in E-Learning', *8th International Conference on Information Visualisation (IV'04)*, IEEE Computer Society, London, UK, pp. 693-700.
- Nguyen, QV & Huang, ML 2004, 'Hierarchical Information Visualization Using ENCCON Model', *IASTED International Conference on Software Engineering (SE 2004)*, IASTED, Innsbruck, Austria, pp. 129-135.
- Nguyen, QV & Huang, ML 2003, 'A Fast Focus+Context Viewing Technique for the Navigation of Classical Hierarchical Layout', *7th International Conference on Information Visualisation (IV'03)*, IEEE, London, UK, pp. 42-46.
- Nguyen, QV & Huang, ML 2003, 'A Fast Focus+Context Viewing Technique for Web Navigation', *International Conference on Internet Programming (IC'03)*, vol. 1, CSREA, Las Vegas, USA, pp. 188-192.
- Nguyen, QV & Huang, ML 2003, 'Visualising File-Systems Using ENCCON Model', *Pan-Sydney Area Workshop on Visual Information Processing (VIP 2003)*, vol 36, Australian Computer Society, Sydney, Australia, pp. 61-66.
- Nguyen, QV & Huang, ML 2002, 'A Space-Optimized Tree Visualization', *IEEE Symposium on Information Visualization 2002 (InfoVis 2002)*, IEEE Computer Society, Boston, Massachussets, USA, pp. 85-92.
- Nguyen, QV & Huang, ML 2002, 'Using Space-Optimized Tree Visualization for Web Site-Mapping', *International Conference on Internet Programming (IC'02)*, CSREA, Las Vegas, USA, pp. 622-628.
- Nguyen, QV & Huang, ML 2002, 'Improvements of Space-Optimized Tree for Visualizing and Manipulating Very Large Hierarchies', *Pan-Sydney Area Workshop on Visual Information Processing (VIP 2002)*, vol. 22, Australian Computer Society, Sydney, Australia, pp. 75-81.

Bibliography

Andrews, K 1995, 'Visualizing Cyberspace: Information Visualization in the Harmony Internet Browser', *IEEE Symposium on Information Visualization*, IEEE Computer Society, USA, pp. 97-104.

Andrews, K & Heidegger, H 1998, 'Information Slices: Visualising and Exploring Large Hierarchies using Cascading, Semi-Circular Discs', *Late-Breaking Hot Topics - IEEE Symposium on Information Visualization (InfoVis'98)*, IEEE, Research Triangle Park, North Carolina, USA, pp. 9-12.

Andrews, K, Wolte, J & Pichler, M 1997, 'Information Pyramids: A New Approach to Visualising Large Hierarchies', *IEEE Visualization'97*, IEEE Computer Society, Phoenix, Arizona, USA, pp. 40-2.

Appelt, W 1999, 'WWW Based Collaboration with the BSCW System', *SOFSEM'99*, vol. 1725 of Lecture Notes in Computer Science, Czech Republic, pp. 66-78.

Bakos, Y 1998, 'The Emerging Role of Electronic Marketplaces on the Internet', *Communications of the ACM*, vol. 41, no. 8, pp. 35-42.

Baldonado, MQW, Woodruff, A & Kuchinsky, A 2000, 'Guidelines for Using Multiple Views in Information Visualization', *Advanced Visual Interfaces (AVI'2000)*, ACM Press, Palermo, Italy, pp. 110-9.

Bederson, BB, Hollan, JD, Perlin, K, Meyer, J, Bacon, D & Furnas, GW 1996, 'Pad++: A Zoomable Graphical Sketchpad For Exploring Alternate Interface Physics', *Journal of Visual Languages and Computing*, vol. 7, no. 1, pp. 3-32.

Bederson, BB, Shneiderman, B & Wattenberg, M 2002, 'Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies', *ACM Transactions on Graphics (TOG)*, vol. 21, no. 4, pp. 833-54.

Biuk-Aghai, RP 1999, 'Visualization of Web-based Workspace Structures', *First International Conference on Web Information Systems Engineering*, IEEE Computer Society, Hong Kong, China, pp. 302-9.

Biuk-Aghai, RP 2001, 'Visualizing Structural and Behavioural Aspects of Virtual Collaboration', *10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001)*, IEEE Computer Society, Cambridge, MA, USA, pp. 279-84.

Brandes, U & Wagner, D 1997, 'A Bayesian Paradigm for Dynamic Graph Layout', *The Symposium on Graph Drawing (GD'97)*, IEEE Computer Society, Rome, Italy, pp. 236-47.

Bruggemann-Klein, A & Wood, D 1988, 'Drawing trees nicely with TEX', *Electronic Publishing*, vol. 2, no. 2, pp. 101-15.

Bruls, M, Huizing, K & van Wijk, JJ 2000, 'Squarified Treemaps', *joint Eurographics and IEEE TCVG Symposium on Visualization*, Springer, Vienna, Austria, pp. 33-42.

Callahan, E & Koenemann, J 2000, 'A comparative usability evaluation of user interfaces for online product catalog', *2nd ACM Conference on Electronic Commerce (EC-00)*, ACM Press, Minneapolis, MN, USA, pp. 197-206.

Card, SK, Mackinlay, JD & Shneiderman, B 1999, *Readings in Information Visualization - Using Vision to Think*, The Morgan Kaufmann series in interactive technologies, Morgan Kaufmann.

Card, SK, Robertson, GG & Mackinlay, JD 1991, 'The Information Visualizer: An Information Workspace', *ACM Conference on Human Factors in Computing Systems (CHI'91)*, ACM Press, New Orleans, USA, pp. 181-8.

Card, SK, Robertson, GG & York, W 1996, 'The WebBook and the Web Forager: An Information Workspace for the World-Wide Web', *CHI'96 Conference on Human Factors in Computing Systems*, ACM Press, Vancouver, BC, Canada, pp. 111-7.

Carpendale, MST, Cowperthwaite, DJ & Fracchia, FD 1997, 'Extending Distortion Viewing from 2D to 3D', *IEEE Computer Graphics & Applications*, vol. 17, no. 4, pp. 42-51.

Carriere, J & Kazman, R 1995, 'Interacting with Huge Hierarchies: Beyond Cone Trees', *IEEE Symposium on Information Visualization (InfoVis'95)*, IEEE Computer Society, Atlanta, Georgia, USA, pp. 74-8.

Cava, RA, Luzzardi, PRG & Freitas, CMDS 2002, 'The Bifocal Tree: a Technique for the Visualization of Hierarchical Information Structures', *Workshop on Human Factors in Computer Systems (IHC2002)*, Fortaleza, Brazil.

Chen, C 1999, *Information Visualization and Virtual Environments*, Springer-Verlag, London.

Cheswick, B & Burch, H 2002, *Internet Mapping Project*, Bell-Labs, viewed 02/07 2002, http://www.cs.bell-labs.com/who/ches/map/index.html.

Cheswick, B, Burch, H & Branigan, S 2000, 'Mapping and Visualizing the Internet', *2000 USENIX Annual Technical Conference*, USENIX, San Diego, CA, USA, pp. 1-12.

Chi, EH 1999, 'Web Analysis Visualization Spreadsheet', *ACM Digital Library Workshop on Organizing Web Space (WOWS '99)*, ACM Press, Berkeley, CA, USA, pp. 24-31.

Chi, EH 2002, *Framework for Visualizing Information*, Kluwer international series on HCI, Kluwer Academic, London.

Chi, EH, Pitkow, J, Mackinlay, JD, Pirolli, P, Gossweiler, R & Card, SK 1998, 'Visualizing the Evolution of Web Ecologies', *CHI'98 Conference on Human Factors in Computing Systems*, ACM Press, Los Angeles, California, USA, pp. 400-7, 644-5.

Chuah, MC 1998, 'Dynamic Aggregation with Circular Visual Designs', *IEEE Symposium on Information Visualization (InfoVis '98)*, IEEE Computer Society, Research Triangle Park, NC, USA, pp. 35-43.

Convertino, G, Chen, J, Yost, B, Ryu, YS & North, C 2003, 'Exploring Context Switching and Cognition in Dual-View Coordinated Visualizations', *Coordinated and Multiple Views In Exploratory Visualization (CMV'03)*, IEEE Computer Society, London, UK, pp. 55-62.

Cox, KC, Eick, SG & He, T 1996, '3D Geographic Network Displays', *SIGMOD Record*, vol. 25, no. 4, pp. 50-4.

Crossley, M, Davies, J, Taylor-Hendry, R & McGrath, A 1997, '3D Internet Developments', *BT Technology Journal*, vol. 15, no. 2, p. 179.

Di Battista, G, Eades, P, Tamassia, R & Tollis, IG 1999, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, New Jersey.

Eades, P 1984, 'A Heuristic for Graph Drawing', *Congressus Numerantium*, vol. 42, pp. 149-60.

Eades, P 1992, 'Drawing Free Trees', *Bulleting of the Institute of Combinatorics and its Applications*, pp. 10-36.

Eades, P & Feng, Q 1996, 'Multilevel Visualization of Clustered Graphs', *Graph Drawing (GD'96)*, vol. 1190, Springer, Berlin, Germany, pp. 101-12.

Eades, P & Huang, ML 2000, 'Navigating Clustered Graphs Using Force-Directed Methods', *Journal of Computational and Graphical Statistics*, vol. 4, no. 3, pp. 157-81.

Eades, P, Lin, T & Lin, X 1993, 'Two Tree Drawing Conventions', *International Journal of Computing Geometry and Applications*, vol. 3, no. 2, pp. 133-53.

Erten, C, Kobourov, SG, Le, V & Navabi, A 2003, 'Simultaneous Graph Drawing: Layout Algorithms and Visualization Schemes', *Graph Drawing (GD 2003)*, vol. 2912, Springer, Perugia, Italy, pp. 98-110.

Feiner, S & Beshers, C 1990, 'Worlds within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds', *User Interface Software and Technology '90*, ACM Press, New York, NY, USA, pp. 76-83.

Feng, Q 1997, 'Algorithms for Drawing Clustered Graphs', The University of Newcastle, Newcastle, Australia.

Friendly, M 2000, *Gallery of Data Visualization - The Best and Worst of Statistical Graphics*, Statistical Consulting Service and Psychology Department York University, viewed 08/08 2004, http://www.math.yorku.ca/SCS/Gallery/.

Furnas, GW 1986, 'Generalized Fisheye Views', *SIGCHI '86 Conference on Human Factors in Computing Systems*, ACM Press, Boston, Massachussetts, pp. 15-22.

Gansner, ER & North, SC 1998, 'Improved Force-Directed Layouts', *Graph Drawing (GD'98)*, Springer, Montréal, Canada, pp. 364-73.

Ghezzi, C 1997, *A geometric approach to three-dimensional graph drawing*, Computation Depatment, UMIST, Manchester, UK.

Hahn, J 2001, 'The Dynamics of Mass Online Marketplaces: A Case Study of an Online Auction', *SIG-CHI on Human factors in computing systems*, ACM Press, Seattle, WA, USA, pp. 317-24.

Hao, MC, Cotting, D, Dayal, U, Machiraju, V & Garg, P 2003, 'Visualizing Multi-Attribute Web Transactions Using A Freeze Technique', *Visualiztion and Data Analysis 2003*, SPIE, Santa Clara, California, USA.

Hawryszkiewycz, I 1999, 'Workspace Networks For Knowledge Sharing', *Fifth Australian World Wide Web Conference (AusWeb99)*, Sydney, Australia, pp. 219-27.

Hendley, RJ, Drew, NS, Wood, AM & Beale, R 1995, 'Narcissus: Visualizing information', *Information Visualization '95 Symposium*, IEEE, Atlanta, GA, USA, pp. 90-6.

Herman, I, Delest, M & Melancon, G 1998, 'Tree Visualisation and Navigation Clues for Information Visualisation', *Computer Graphics Forum*, vol. 17, no. 2, pp. 153-65.

Herman, I, Melancon, G, de Ruiter, MM & Delest, M 1999, 'Latour - a Tree Visualisation System', *Symposium on Graph Drawing (GD'99)*, Springer-Verlag, Stirin Castle, Czech Republic, pp. 392-9.

Herman, I, Melancon, G & Marshall, MS 2000, 'Graph Visualization in Information Visualization: a Survey', *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, pp. 24-44.

Hollands, JG, Carey, TT, Matthews, ML & McCann, CA 1989, 'Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views', *Designing and Using Human Computer Interfaces and Knowledge Based Systems*, Amsterdam, pp. 313-20.

Huang, ML & Eades, P 1998, 'A Fully Animated Interactive System for Clustering and Navigating Huge Graphs', *Graph Drawing (GD'98)*, Springer, Montréal, Canada, pp. 374-83.

Huang, ML, Eades, P & Cohen, RF 1998, 'On-line Animated Visualization of Huge Graphs Using a Modified Spring Algorithm', *Journal of Visual Languages and Computing*, vol. 9, no. 6, pp. 623-45.

Huang, ML & Zhang, K 2002, 'Navigating Product Catalogs Through OFDAV Graph Visualization', *International Conference on Distributed Multimedia Systems (DMS'2002)*, Knowledge Systems Institute, Redwood City, CA, USA, pp. 555-61.

Inxight 2004, *Hyperbolic Browser (Start Tree) for Online Shopping Stores*, viewed 11/03 2004, http://www.inxight.com.

Jankun-Kelly, TJ & Ma, K-L 2003, 'MoireGraphs: Radial Focus+Context Visualization and Interaction for Graphs with Visual Nodes', *IEEE Symposium on Information Visualization 2003*, IEEE Computer Society, Seattle, Washington, USA, pp. 59-66.

Jog, NK & Shneiderman, B 1995, 'Starfield Visualization with Interactive Smooth Zooming', *Visual Database Systems 3, Visual Information Management (VBD'95)*, vol. 34, Chapman & Hall, Lausanne, Switzerland, pp. 3-14.

Johnson, B & Shneiderman, B 1991, 'Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures', *the 1991 IEEE Visualization*, IEEE Computer Society, Piscataway, NJ, USA, pp. 284-91.

Kadmon, N & Shlomi, E 1978, 'A Polyfocal Projection for Statistical Surfaces', *the Cartographic Journal*, vol. 15, no. 1, pp. 36-41.

Kennedy, AJ 1996, 'Functional Pearls - Drawing Trees', *Journal of Functional Programming*, vol. 6, no. 3, pp. 527-34.

Kim, J 1999, 'An Empirical Study of Navigation Aids in Customer Interfaces', *Behaviour & Information Technology*, vol. 18, no. 3, pp. 213-24.

Kleiberg, E, van de Wetering, H & van Wijk, JJ 2001, 'Botanical Visualization of Huge Hierarchies', *IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, IEEE Computer Society, San Diego, CA, USA, pp. 87-94.

Kobourov, SG & Wampler, K 2004, 'Non_Euclidean Spring Embedders', *IEEE Symposium on Information Visualization 2004*, IEEE, Austin, Texas, USA, pp. 207-14.

Kreuseler, M & Schumann, H 1999, 'Information Visualization Using a New Focus+Context Technique in Combination with Dynamic Clustering of Information Space', *Workshop on New Paradigms in Information Visualization and Manipulation (NPIVM '99), in conjunction with 8th ACM international Conference on Information and Knowledge Management (CIKM '99)*, ACM Press, Kansas City, Missouri, USA, pp. 1-5.

Lamping, J & Rao, R 1996, 'The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies', *Journal of Visual Languages and Computing*, vol. 7, no. 1, pp. 33-55.

Lee, J, Lee, HS & Wang, P 2001, 'Design and Implementation of a Visual Online Product Catalog Interface', *3rd International Conference on Enterprise Information Systems*, vol. 2, Setubal, Portugal, pp. 1010-7.

Leung, YK 1989, 'Human-computer interaction techniques for map-based diagrams', *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Amsterdam, pp. 361-8.

Leung, YK & Apperley, MD 1994, 'A Review and Taxonomy of Distortion-Oriented Presentation Techniques', *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 1, no. 2, pp. 126-60.

Mackinlay, JD, Robertson, GG & Card, SK 1991, 'The Perspective Wall: Detail and Context Smoothly Integrated', *ACM Computer-Human Interaction '91 Conference on Human Factors in Computing Systems*, ACM Press, New York, USA, pp. 173-9.

McGuffin, MJ, Davison, G & Balakrishnan, R 2004, 'Expand-Ahead: A Space-Filling Strategy for Browsing Trees', *IEEE Symposium on Information Visualization 2004*, IEEE, Austin, Texas, USA, pp. 119-26.

Melancon, G & Herman, I 1998, *Circular Drawings of Rooted Trees*, Technical Report: INS-R9817, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands.

Mitta, DA 1990, 'A Fisheye Presentation Strategy: Aircraft Maintenance Data', *INTERACT '90*, IFIP, Elsevier, pp. 875-80.

Munzner, T 1997, 'Exploring Large Graphs in 3D Hyperbolic Space', *IEEE Comp. Graphics & Applications*, vol. 18, no. 4, pp. 18-23.

Neumann, A 2000, 'A Better Mousetrap Catalog', *Business 2.0*, pp. 117-8.

North, C 2001, 'Multiple Views and Tight Coupling in Visualization: A Language, Taxonomy, and System', *The 2004 International Multiconference in Computer Science and Computer Engineering*, CSREA, Las Vegas, Nevada, USA, pp. 626-32.

North, C & Shneiderman, B 2000, 'Snap-Together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata', *Advanced Visual Interfaces*, Palermo, Italy, pp. 128-35.

North, C & Shneiderman, B 2001, 'Component-Based, User-Constructed, Multiple-View Visualization', *CHI 2001 Conference on Human Factors in Computing Systems*, ACM Press, Seattle, Washington, USA, pp. 201-2.

North, SC 1995, 'Incremental Layout in DynaDAG', *Graph Drawing (GD '95)*, Springer, Passau, Germany, pp. 409-18.

Osawa, N 2001, 'A Multiple-Focus Graph Browsing Technique Using Heat Models and Force-Directed Layout', *Fifth International Conference on Information Visualisation (IV'01)*, IEEE Computer Society, London, UK, pp. 277-83.

Perlin, K & Fox, D 1993, 'Pad: An Alternative Approach to the Computer Interface', *20th International Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH '93)*, ACM Press, New York, USA, pp. 57-64.

Plaisant, C, Grosjean, J & Bederson, BB 2002, 'SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation', *IEEE Symposium on Information Visualization (InfoVis 2002)*, IEEE, Boston, MA, USA, pp. 57-64.

Reingold, EM & Tilford, JS 1981, 'Tidier Drawing of Trees', *IEEE Transactions on Software Engineering*, vol. 7, no. 2, pp. 223-8.

Rekimoto, J & Green, M 1993, 'The information cube: Using transparency in 3D information visualization', *Third Annual Workshop on Information Technologies & Systems (WITS'93)*, pp. 125-32.

Robert, JC 1998, 'On Encouraging Multiple Views for Visualisation', *International Conference on Information Visualisation (IV 1998)*, IEEE Computer Society, London, UK, pp. 8-14.

Robert, JC 2000, 'Multiple-View and Multiform Visualization', *Visual Data Exploration and Analysis VII*, vol. 3960, IS&T and SPIE, pp. 176-85.

Robertson, GG, Card, SK & Mackinlay, JD 1989, 'The Cognitive Co-processer for Interactive User Interfaces', *ACM Symposium on User Interface Software and Technology*, ACM Press, pp. 10-8.

Robertson, GG, Mackinlay, JD & Card, SK 1991, 'Cone Tree: Animated 3D Visualizations of Hierarchical Information', *ACM SIGCHI conference on Human Factors in Computing Systems '91*, ACM Press, New York, USA, pp. 189-94.

Roseman, M & Greenberg, S 1996, 'TeamRooms: Network Places for Collaboration', *ACM Conference on Computer-Supported Cooperative Work (CSCW'96)*, ACM, pp. 325-33.

Roth, SF, Chuah, MC, Kerpedjiev, S, Kolojejchick, JA & Lucas, P 1997, 'Towards an Information Visualization Workspace: Combining Multiple Means of Expression', *Human-Computer Interaction Journal*, vol. 12, no. 1&2, pp. 131-85.

Sarkar, M & Brown, MH 1994, 'Graphical Fisheye Views', *Communications of the ACM (CACM)*, vol. 37, no. 12, pp. 73-84.

Shaw, W & Tigg, J 1993, *Applied Mathematica: Getting Started, Getting It Done*, Pearson Higher Education.

Shiloach, Y 1976, 'Arrangements of Planar Graphs on the Planar Lattices', Ph.D Thesis thesis, Weizmann Institute of Science, Rehovot, Israel.

Spence, R 2001, *Information Visualization*, Addison-Wesley, Harlow, England.

Spence, R & Apperley, MD 1982, 'A Bifocal Display Technique for Data Presentation', *of Eurographics 82*, pp. 27-43.

Stasko, J & Zhang, E 2000, *Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualization*, Report Number GIT-GVU-00-12, GVU Center and College f Computing, Georgia Institute of Technology, Atlanta, USA.

Teoh, ST & Ma, K-L 2002, 'RINGS: A Technique for Visualizing Large Hierarchies', *Graph Drawing 2002*, Springer, Irvine, California, USA, pp. 268-75.

van Wijk, JJ & Nuij, WAA 2003, 'Smooth and Efficient Zooming and Panning', *IEEE Symposium on Information Visualization 2003*, IEEE, Seattle, Washington, USA, pp. 15-22.

van Wijk, JJ & van de Wetering, H 1999, 'Cushion Treemaps: Visualization of Hierarchical Information', *IEEE Symposium on Information Visualization 1999 (INFOVIS'99)*, IEEE Computer Society, San Francisco, California, USA, pp. 73-8.

Ware, C 2000, *Information Visualization: Perception For Design*, ed. TMKSiI Technolgies, Morgan Kaufmann.

Ware, C 2004, *Information Visualization: Perception for Design*, Second edn, Morgan Kaufmann, San Francisco, CA.

Ware, C, C. Purchase, H, Colpoys, L & McGill, M 2002, 'Cognitive Measurements of Graph Aesthetics', *Information Visualization Journal (Palgrave Macmillan)*, vol. 1, no. 2, pp. 103-10, viewed June.

Wilkins, AJ 1995, *Visual Stress*, Oxford Psychology Series #24, Oxford University Press.

Wills, GJ 1999, 'NicheWorks - Interactive Visualization of Very Large Graphs', *Journal of Computational and Graphical Statistics*, vol. 8, no. 2, pp. 190-212.