Structural Focus + Context Navigation of Relational Data

by

Kevin James Pulo

Bachelor of Computer Science (Honours, Advanced), 2000

A thesis submitted to
The School of Information Technologies
The University of Sydney
for the degree of
DOCTOR OF PHILOSOPHY

October 2004

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

_____

Kevin James Pulo

Sydney
25th July 2005

For Mum and Dad;
without whom this would not have been possible.

Acknowledgements

This work would not have been possible if it were not for the generous help and assistance that I have received from numerous people. First, my supervisor Peter Eades, for his constant guidance, enthusiasm and prodding throughout the course of my PhD — particularly when I was unsure of my topic. Thanks also to Tom Sawyer Software for supporting my research with a top-up scholarship, and in particular to Francois Bertault, Wendy Feng, Brendan Madden and Andrew Tom. I am indebted to Geoff Dromey and his research group at Griffith University, for providing me with Design Behaviour Trees and guiding me in their operation — in particular, to the students who generously donated their time to create and contribute logfiles. I am also grateful to Keith Nesbitt for the use of his Multi-Sensory Taxonomy from his PhD thesis as an additional data source, and to Andrew Macks, for relieving me of the relatively mundane work involved in crawling and parsing the ACM citation data. Thanks to all those who hosted me in their research groups on my travels, and the thoughtful discussions that ensued: TSS, UNH, AT&T, MERL and SDSC. Particular appreciation to Jim Lundholm-Eades and family in Minnesota, and to Karen Potter and David Ellis in London, all of whom went out of their way to show me nothing but the utmost hospitality. I am appreciative of Paul Mutton for the use of his excellent EPSGraphics2D package, which is responsible for the high-quality vector screenshots in this thesis; and to Honza Holecek for his work in embedding native animations in PDF files, which forms the basis of the many animated figures in this thesis. Thanks to all the members of the Infovis and HUM research groups who provided many stimulating discussions and excellent friendship, in particular, to Tim Dwyer for proofing part of my thesis, introducing us to espresso coffee, and beating me to submission. Much thanks to Dr. St. Clair-Kendall for proof-reading the entire manuscript. Finally, I would like to dedicate special mention to Mum, Dad, Jarod and Smudge, who always provided continuous love and have supported me in all forms throughout the course of my education.

Contents

Abstract

Traditional information visualisation is concerned with methods of drawing a complete picture of an entire dataset. By contrast, much of modern information visualisation deals with the problem of how to see datasets that are too large to be displayed *in toto*. This problem is known as *large scale* information visualisation. The most common approach is to display only part of the dataset, but allow the user to *navigate* easily to other parts of the dataset that are not shown. *Focus + Context* techniques address large scale information visualisation by presenting a small amount of "focus" data at a high level of detail, surrounded by the majority of the remaining data at a low level of detail, the "context". The majority of Focus + Context techniques to date have been based on *geometric distortion*, where the visualisation of the entire dataset is adjusted to show the focus region at normal magnification, whilst demagnifying the context region.

An alternative to geometric distortion is *data-driven* Focus + Context, where the concepts of "focus", "context", "zooming" and "navigation" are defined in terms of *multi-detail* datasets that store the data using multiple levels of detail. Data-driven methods require the development of new techniques for the presentation and animation of information. This thesis presents a new data-driven Focus + Context technique which we call *Structural Zooming*.

Structural Zooming is presented in a visualisation independant way that allows any illustration of any type of data to be adapted for use with Structural Zooming. Further, a method is given for performing Structural Zooming of relational data, namely trees and clustered graphs. This has the advantages of geometric zooming techniques (such as Graphical Fisheye Views and the Hyperbolic Browser), including high detail focus, low detail context, smoothly animated transitions during navigation and preservation of a high quality, aesthetically pleasing layout. In addition, it has advantages over geometric zooming, including an approximately constant level of visual complexity by presenting less data at lower detail in the context region, preservation of spatial properties and the ability to leverage existing information visualisation techniques. We define empirical quality measures and present an experimental evaluation of Structural Zooming of relational data using these measures. This evaluation utilises a corpus of data files from three application domains, and navigation data derived both from real users and computational models of navigation, in order to validate the design choices made in the application of Structural Zooming to relational data.

CHAPTER 1

Introduction

Black then white are, all I see, in my infancy,
red and yellow then came to be, reaching out to me,
lets me see. There is so much more that beckons me,
to look through to these infinite possibilities.
As below, so above and beyond, I imagine.
Drawn outside the lines of reason.
Push the envelope. Watch it bend.

– – Maynard James Keenan, "Lateralis"

The field of information visualisation is concerned with effective ways to see very large amounts of data. Vast amounts of data of many types is being collected at rapid rates in many areas of science and business [1, 8]. Similarly, the capacity of mass online storage devices is rapidly increasing [68]. In many cases, this data is being collected and archived faster than it can be processed or analysed by humans or machines. Often this data is so large that specialised new algorithms known as *external memory algorithms* are required [2, 3], since the efficiency of existing algorithms hinges on loading the entire dataset into memory. In order to be of any real use, this data must be processed in a timely manner, and it must be possible for humans to obtain actual information from the data. This means that the data must be *displayed* in some way to human users. Figure 1.1 shows the flow of data, from its collection in the real world, to storage as data, its display on a display device, and finally to perception by a human. At each of these stages, the available bandwidth and processing power diminishes, as indicated by the arrow sizes. Information visualisation is concerned with the final two stages, that of presenting data on a display device that is subsequently viewed by humans.

There are two main ways to approach the visualisation of data, illustrated in Figure 1.2. The first is to directly visualise the data, presenting all or part of the data to the user. The second is to

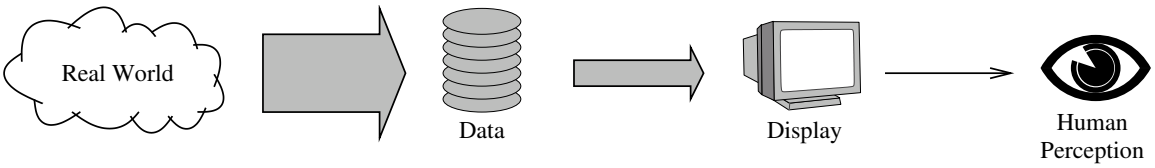


Figure 1.1: The flow of data from the real world to humans through information visualisation.

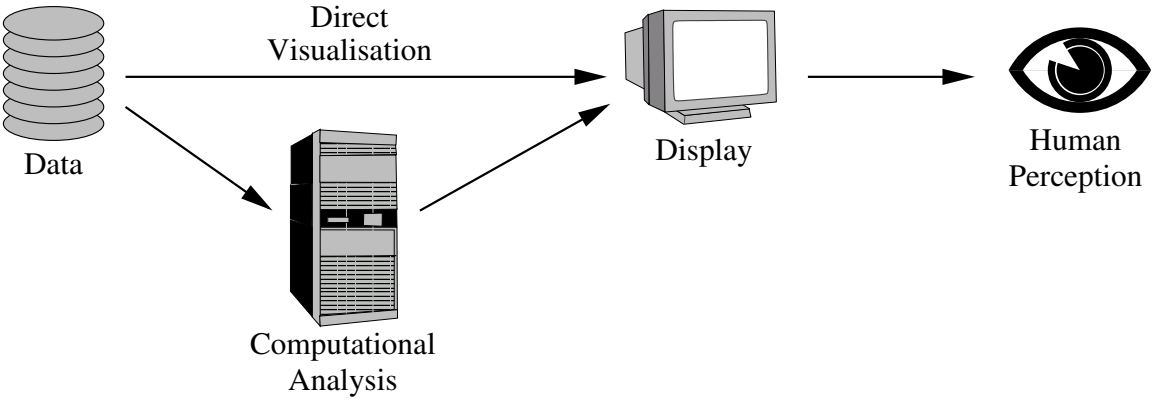


Figure 1.2: The two main approaches for the visualisation of data.

use *computational analysis* to process the data in some way (for example, *data mining* techniques such as *clustering*), and then visualise the results of this processing. Such strategies are often difficult to implement and suffer from scalability and complexity issues, requiring inordinately large amounts of computational power (such as *distributed* and *parallel supercomputers*). In addition, the presentation of the algorithm results to the user is also a visualisation problem. For these reasons, this thesis deals with the direct visualisation of data, where the data may be original raw data, or may be the results of various stages of processing.

The idea behind direct visualisation is to utilise the potential processing power of the human brain. The brain is very good at processing certain types of information, such as visual patterns, and can do these tasks very quickly and with high accuracy [166]. Many perceptual tasks are performed automatically at a low-level, which is not only fast, but frees the higher level cognitive functions to concentrate on harder problems, such as understanding and interpreting the data, and making decisions based on it. Presenting the data to the user in the right way can allow the user to "see" more information in the data more quickly than any amount of computational power. This is an area of information visualisation known as *visual data mining*, so-called because the "data mining" is performed visually by the human [82]. However, not all visualisations have this effect. Because the

vision system is so influential in most human thought processes, it is easy for erroneously presented data to be highly confusing and even misleading (either intentionally or accidentally) [159]. Thus, it is important to ensure that all visualisations are of high quality.

This thesis is concerned with presenting large amounts of data to human users in ways that facilitate these benefits. The aim is to allow users to view, navigate and potentially modify the large dataset with the same level of ease afforded by smaller, more accessible datasets.

1.1 The detail-context tradeoff

The fields of computer graphics and human-computer interaction received a major advance in 1962 with the development of *Sketchpad* [153, 154], a revolutionary computer aided drawing program. Sketchpad used a *cathode ray tube* as a *vector* display device to show geometric objects as a collection of smooth lines and curves. However, all modern computer displays present images in a *raster* form, that is, as a 2 dimensional matrix of small display elements known as *pixels* (short for *picture elements*). The transition from vector to raster presentations was mainly due to memory becoming available in the quantities and speeds required to support a *frame buffer* of pixels to be shown. Each pixel in a raster display can assume a colour, and so an image is created by choosing pixel colours appropriately. The number of pixels in a display is its *resolution* and the number of possible colours for each pixel is its *colour depth*. In 2004, a typical computer display has resolution between 1024×768 (about 750 000 pixels, or 0.75 *megapixels*) and 1600×1200 (about 1.92 megapixels), and colour depth around 2^{16}.

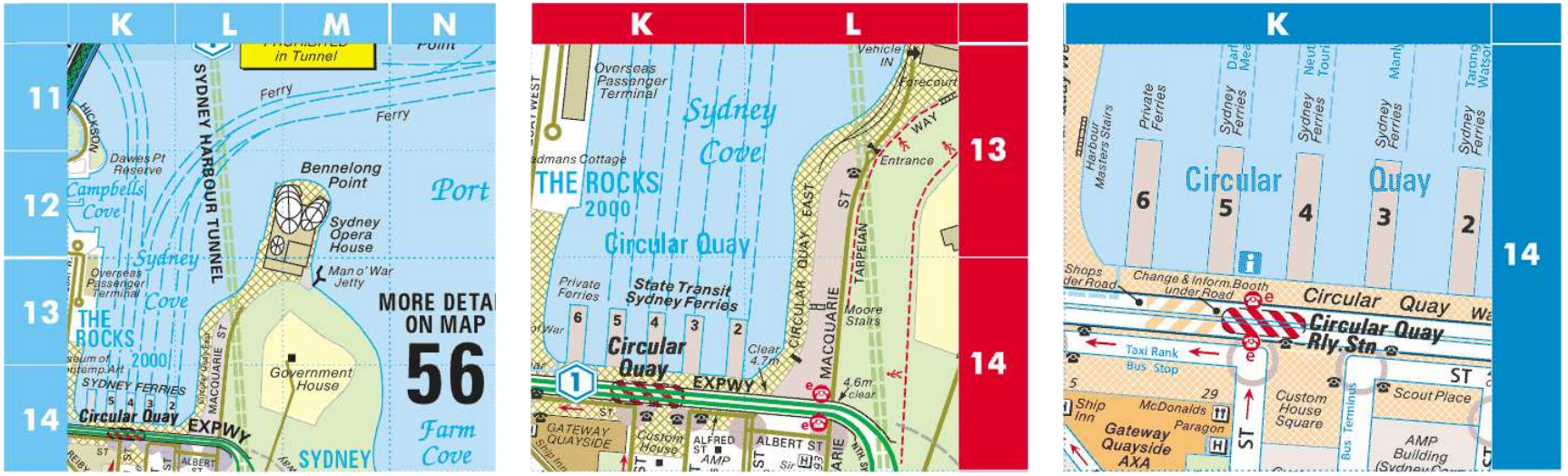
The quality of a raster display depends directly on its resolution and colour depth. Specifically, an increase in resolution or colour depth corresponds to an increase in the amount of fine detail that can be shown in the image. However, colour depth is of more importance for photographic and photorealistic images; information visualisation is only rarely limited by the colour depth available on modern computers [107]. On the other hand, resolution is increasingly a limiting factor in information visualisation. The resolution of standard desktop monitors has remained between about 1 and 2 megapixels for the past 10 years, and projector displays have been limited to about 1 megapixel during this time. This is an issue, when coupled with the need to visualise large amounts of data. Advances in seamless tiled display technology such as the 9 megapixel High Density Display at the San Diego Supercomputer Center [134] have helped to increase the available resolution, but require highly specialised and bulky hardware and careful calibration to achieve acceptable image quality.

Desktop monitors with resolutions as high as 9 megapixels (3840×2400) were announced in 2004 [171], although at a very high price. Although this may seem like a large increase (almost a factor of 10), the growth of dataset sizes is expected to easily outstrip any foreseeable improvements in display resolution. In addition, there are fundamental limits of human perception that cannot easily be improved. These low-level limits approximately correspond to a 16 megapixel display device [166], and are discussed in more detail below. For these reasons, much research in information visualisation has focused on methods of presenting data on displays of relatively poor resolution, compared to the size of the data itself. For example, displaying a tree or graph with 10,000 labelled nodes on a 1 megapixel display allows only 100 pixels per node on average — not even enough to show a node label of a moderate amount of text. This means that a more sophisticated method of visualisation is required.

The assumption of a fixed display resolution gives rise to the *detail–context tradeoff*. Display space in the form of pixels may be used either to show a large amount of detail for a small logical area, or to show a low level of detail for a large logical area. Figure 1.3 demonstrates this idea by showing a section of a street map for Circular Quay in Sydney Harbour at a fixed resolution. Figure 1.3(a) shows a large area at low detail, Figure 1.3(b) shows a smaller area at an intermediate higher level of detail, and Figure 1.3(c) shows a small area at a high level of detail. Many fine details, such as the information booth between wharves 4 and 5, are present in Figure 1.3(c) that would be impossible to show in Figure 1.3(a). However, Figure 1.3(c) is lacking in *context* information that indicates locations on a larger scale. For example, it is not possible to see surrounding monuments in Figure 1.3(c), whereas Figure 1.3(a) clearly shows the Sydney Opera House and its location relative to Circular Quay. Figure 1.4 illustrates the detail–context tradeoff diagrammatically. The exact position of the line depends on the characteristics of the display.

The human visual system has been extensively studied in the area of *perceptual psychology*, both in general [65, 140] and as it relates to information visualisation [166]. This research suggests that the "bandwidth" of the human perceptual system for gathering and processing sensory information is limited [166]. This limitation is a result of human physiology and psychology, which is, for the most part, "hard-wired" into our brains such that we have little or no direct control over the systems involved.

For example, the number, density and arrangement of cone receptors on the retina of the eye, and the "brain pixels" associated with ganglion structures, are physiological limitations over which



(a) Lowest detail (1:20000 scale)

$\text{Area} = 1000\text{m} \times 1000\text{m} = 100 \times 10^4\text{m}^2$

$\text{Pixel area} = 2.54\text{m} \times 2.54\text{m} = 6.44\text{m}^2$

$\text{Pixel density} = 0.16 \text{ pixels}/\text{m}^2$

(b) Medium detail (1:10000 scale)

$\text{Area} = 500\text{m} \times 500\text{m} = 25 \times 10^4\text{m}^2$

$\text{Pixel area} = 1.27\text{m} \times 1.27\text{m} = 1.61\text{m}^2$

$\text{Pixel density} = 0.62 \text{ pixels}/\text{m}^2$

(c) Highest detail (1:5000 scale)

$\text{Area} = 250\text{m} \times 250\text{m} = 6.25 \times 10^4\text{m}^2$

$\text{Pixel area} = 0.63\text{m} \times 0.63\text{m} = 0.40\text{m}^2$

$\text{Pixel density} = 2.48 \text{ pixels}/\text{m}^2$

Figure 1.3: An example of the detail–context tradeoff in the area of street maps. Each map shows Circular Quay (map reference K-14) in Sydney, Australia, as a $394 \times 394 = 155\,236$ pixel image. Each grid reference square represents $250\text{m} \times 250\text{m}$. Copyright Sydway Publishing 2003. Reproduced from Sydway Greater Sydney Street Directory Edition 9 with permission.

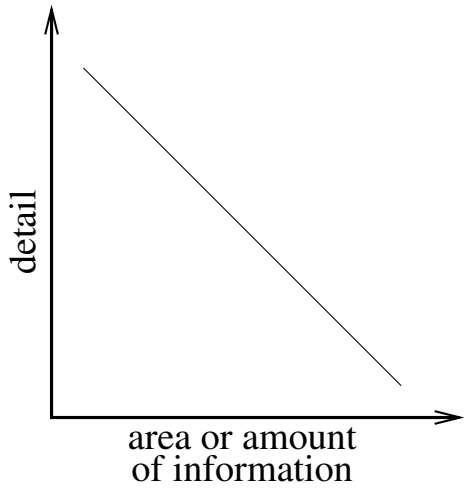


Figure 1.4: The detail–context tradeoff.

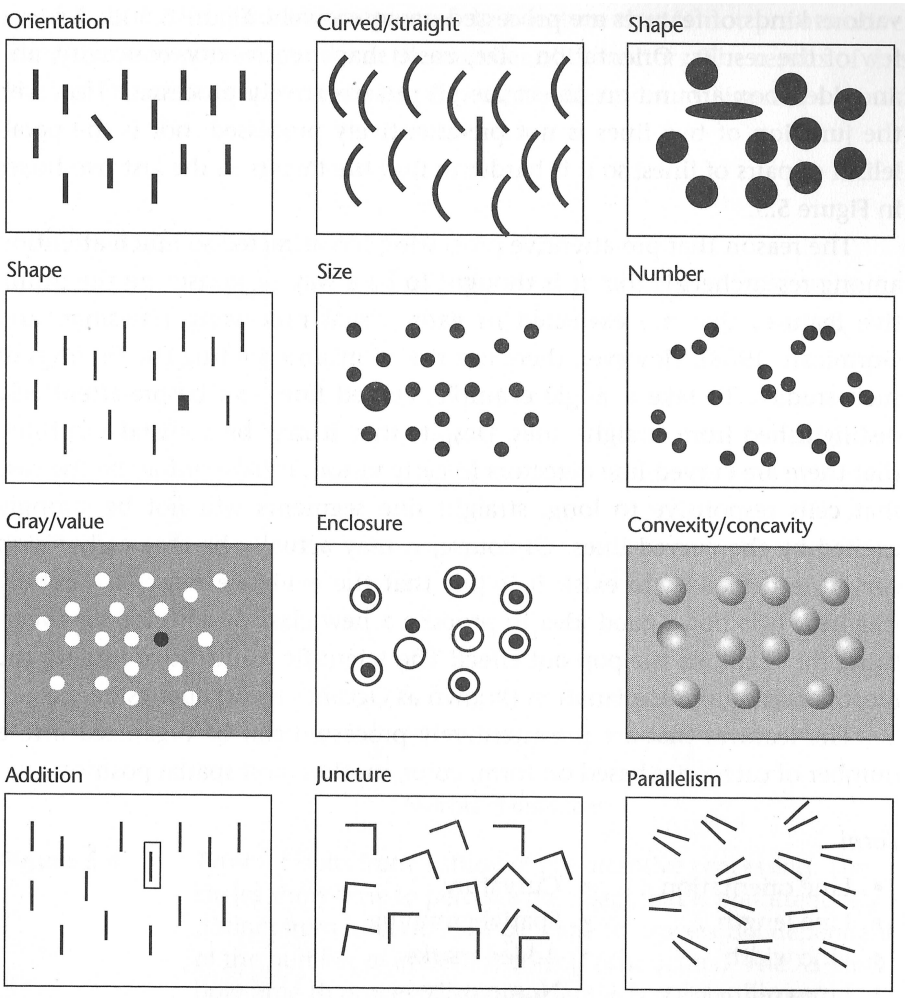


Figure 1.5: From [166] (Figure 5.5): Most of the differences shown above are pre-attentively distinguished. Only juncture and parallellism are not. Reprinted from [166], Copyright 2000, with permission from Elsevier.

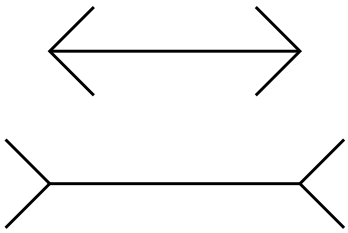


Figure 1.6: The Müller-Lyer optical illusion.

humans have little control. In the context of information visualisation this places practical limits on the necessary spatial and colour resolution of display devices.

An example of a psychological limitation is the way in which the human visual system perceives certain visual stimuli *pre-attentively*, that is, involuntarily and without conscious awareness of it. Pre-attentive processing is very fast and powerful, as it allows particular types of visual stimuli to be identified and located in an amount of time that is independant of the number of "distractor" objects. For patterns that are not pre-attentively processed, a linear search through all the objects must be used [166]. Figure 1.5 presents a collection of images that show various pre-attentive attributes. In the context of information visualisation, this suggests that these attributes may be exploited for applications involving visual data mining or visual search. A related area is *optical illusions* [65, 71, 139], such as the classic Müller-Lyer illusion shown in Figure 1.6 where the apparent lengths of line-segments is influenced by other lines at the endpoints. Such misleading illusory effects should be rigorously avoided in information visualisation.

An example of a cognitive limitation is the number of "chunks" of information that humans can retain in short-term memory (also known as "working memory"). This well-known result states that humans are able to reason most effectively with 7 ± 2 "units of information" stored in short-term memory [100]. In the context of information visualisation, this suggests that the amount of information that can be recalled between views in an interactive navigation is limited to around 7.

Thus, although display device technology is improving, the fundamental limitations in the human visual perception system mean that the available "bandwidth" for the visual presentation of information is limited. Increasing these limitations is not possible, and so information visualisation must devise methods that work well in these limited conditions.

Large scale information visualisation is concerned with techniques for the visualisation of large amounts of data, specifically, more than can fit on commonplace display devices using traditional

techniques. In this situation, the detail–context tradeoff is a particular problem. This is because the detail required to display an amount of data on the screen increases as the size of the data increases, and thus the lower the context that may be displayed.

The traditional and naïve methodology for dealing with this is known as *Simple Pan + Zoom*. This displays the data at a particular *uniform* zoom level over the entire display, and provides the user with navigation operations for *panning* and changing the zoom level. Panning is moving to neighbouring or adjacent regions in the data without changing the zoom level. This methodology is only acceptable for small datasets, and does not scale up well to large scale information visualisation. This is because when 'zoomed in' it provides no context, and when 'zoomed out' it provides no details. Thus, the user is required to navigate between such views and combine them mentally. A much more desirable system is one in which some context information can be provided in addition to a detailed view, these are known as "Focus + Context" systems, and are discussed below.

Interactive navigation is an essential part of all large scale information visualisation systems, in order that users may see information that is not currently shown. This requires users to mentally integrate temporally disjoint views of the data. Research has shown that humans do this by forming an internalised "mental map" or "cognitive map" of the data [87, 88, 157, 162]. As the views are temporally disjoint, users must rely on their limited short term memory when comparing two or more views, thus potentially hindering the construction of the mental map.

Whilst pyschological researchers are in agreement that such a map exists, there is no concensus on the exact nature of how these maps are stored and processed by the brain. However, the fact that there is some spatial or visual basis to the mental map is apparent from the way that a user's mental map can be "lost", or made useless, by rearranging the contents of a diagram. This has been studied in the field of graph drawing [43, 101], where three mathematical models are presented for the preservation of the mental map between subsequent views of a diagram. These mathematical models are:

Orthogonal Ordering is the simplest and most basic representation of the mental map. It states that the "up", "down", "left" and "right" relations between all pairs of nodes should be maintained between layouts of a graph.

Proximity Relations is the idea that the distances between nodes ought to be preserved: close nodes should stay nearby, and far nodes should stay further

away. This can also be thought of as the spatial clustering of the nodes in the diagram.

Topology is concerned with the faces of the plane that are created by the edges (and their crossings) in a graph. In graph drawing, this is known as the *embedding* of the graph, and is quantified by the *dual graph* (where edge crossings are considered to be the insertion of a "dummy" node, causing this modified graph to have the property of being straight-line planar).

The *Force Scan Algorithm* (*FSA*) [42, 89, 90, 101] applies these mental map principles to the problem of creating disjoint-node images from the output of classical graph drawing algorithms. It is discussed in more detail in Section 4.2. The SHriMP Layout Adjustment Algorithm [149] is an application of these mental map principles to the hierarchical Geometric Fisheye View found in the SHriMP visualisation system, which is discussed further in Section 1.2.3 below. Animation has been suggested as a method of improving the construction and preservation of the mental map; this is discussed in more detail in Section 1.2.5 below.

This thesis utilises these mental map models in two places: in the design of a "stable" inclusion tree layout, presented in Section 4.2.1, and in the design of several measures for the empirical evaluation of Structural Zooming of relational information, presented in Section 4.4.

The *Focus + Context* methodology of large scale information visualisation attempts to resolve the problems of Simple Pan + Zoom by displaying a small amount of data at high detail, known as the *focus view*, together with a large amount of the data at low detail, known as the *context view*. This has the benefit of allowing the user to see details in the region of interest, whilst still seeing the approximate location of this focus data in relation to the entire dataset. This transforms the task of finding and examining context, in order to support navigation, from being cognitive to being perceptual, and supports the creation and preservation of the user's mental map.

1.2 Previous work

This section reviews previous work in areas related to large scale information visualisation, the mental map and animation.

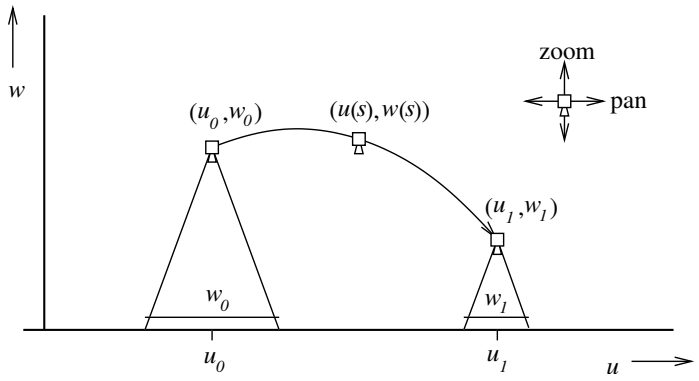


Figure 1.7: A (u, w) space-scale diagram from [165]. The w axis corresponds to the distance of the camera above the plane, and is thus the "scale" of the produced image. The u axis corresponds to space, here it is shown as 1D, but can easily be extended to 2D (consider the u axis to be a "side" view of the plane).

1.2.1 Uniform zooming

Uniform zooming is a class of visualisation techniques where a uniform geometric magnification is applied to the display. It includes Simple Pan + Zoom techinques, as well as various extensions that are now discussed.

Rapid zooming, also known as the *zoomable user interface* (or *ZUI*), aims to reduce the cognitive effort associated with navigating in the absence of any on-screen context. It works by considering the visualisation to exist on an infinite plane of infinite resolution. The view is that obtained by a virtual camera viewing the plane perpendicularly from above. Zooming in and out corresponds to the distance of the camera above the plane, as shown in the *space-scale diagram* in Figure 1.7 [61].

Rapid zooming methods and ZUIs allow the user to be able to zoom in and out of the scene very rapidly, and these operations are animated in the manner that would be observed whilst moving the camera vertically. This allows the user to very quickly change between an overview of the visualisation and any detailed view. The idea is that the rapidity of this operation lowers the cognitive barrier to the user's understanding of the entire visualisation. Several software toolkits are available that allow ZUIs to be easily developed and integrated into other information visualisation systems — the most notable being the Pad—Pad++—Jazz—Piccolo lineage of toolkits [19, 22].

In uniform zooming systems, panning operations may additionally make use of the zoom level, rather than being at a constant zoom level as in Simple Pan + Zoom. A straightforward strategy is to have the system automatically zoom out to a full overview of the visualisation, following by zooming in to the new detail location. This allows the user to see the context of the two detail views. However, it may not be necessary to zoom out completely, and it is possible to begin panning prior

to the end of the zoom-out operation. This corresponds to a path in the space-scale diagram, such as from (u_0, w_0) to (u_1, w_1) in Figure 1.7. Research on such *speed-dependant automatic zooming* has shown it to be superior to conventional Simple Pan + Zoom (which is the straight line between (u_0, w_0) and (u_1, w_1) in Figure 1.7) [27, 78, 165, 168].

Semantic zooming is a related technique for automatically adjusting the visual representation of the displayed data, depending on the zoom level [58, 59, 116]. For example, a map of a railway network when zoomed-out shows only circles for stations and connecting lines. As the user zooms-in to a particular region, the stations may change to include a circle and the name of the station. Zooming in further may change the station representation to indicate its shape and number of platforms. The highest level of zoom, when the user is zooming-in on one station alone, may be an arial photograph or floorplan of the station, showing entrances, exits, facilities, and so on. The benefit of semantic zooming, though, is that it offers the possibility of changing the representation to show other information, rather than the simple geometric example described above. For example, when a user zooms-in to a single station the consequent display may show timetable or fare pricing information relating to that station.

Uniform zooming systems generally suffer from problems relating to the *density* of information presented. This is an issue in the zooming axis: different zoom levels may show different amounts of information. It is also an issue in the spatial dimension: different parts of the visualisation may have different amounts of information. These problems are addressed in research on *constant information density* [173, 174], which presents a method for equalising the density of information shown in rapid zooming and semantic zooming systems. This work has not been applied to geometric Focus + Context systems (described below in Section 1.2.3), although doing so should be straightforward.

1.2.2 Multiple coordinated views

A natural extension of uniform zooming is that of *multiple coordinated views* for presenting data at several disjoint levels of detail. The *Overview + Detail* method [118] is commonly used, due to its simplicity. It is similar to the Simple Pan + Zoom techniques, with the addition of an *overview window* that shows a highly demagnified, low detail version of the entire visualisation. The overview window may or may not obscure part of the detail window. A visual cue, usually in the form of a rectangle, is shown on the overview to indicate the location of the detail view. This provides orientation in the same way as “You are here” markers on maps. This cue may be strengthened with the use of *tethers* joining the corners of the detail window with the corners of its rectangle in the

overview window. For example, Figure 1.8 shows a satellite image of the city of San Francisco, with a detailed view of the area near Union Square. The user is able to pan around by clicking in the overview window.

The DragMag system [170], as shown in Figure 1.9, uses an *Inverted* Overview + Detail view, showing the overview window occupying the whole screen, with one or more tethered detail-windows. As with normal Overview + Detail, the user is able to navigate by clicking or dragging in the overview-window to change the detail view. Some systems place the detail-window directly over its location in the overview window, by taking the metaphor of holding a magnifying glass over the visualisation. However, this strategy causes the detail-window to obscure the context in its immediate vicinity, such as shown in Figure 1.10. For this reason, the DragMag system deliberately places detail-windows a small distance away from their location in the overview-window.

Recent work has further developed the idea of tethered detail windows, particularly in 3D virtual environments such as the GeoZUI3D browser shown in Figure 1.11. [119, 120, 121].

1.2.3 Geometric focus + context

Geometric Focus + Context is a class of visualisation techniques that show focus and context regions by displaying different regions of the visualisation at varying *magnification factors*. This causes the visualisation to be *distorted*: the focus-region has a magnification of 1 (that is, it is presented at its normal size), while the context-region typically has a magnification that approaches 0 as the distance from the focus-region increases (that is, it is progressively demagnified). This concept is formally specified through a *transformation function* or *magnification function*. These two functions are different, equivalent ways of expressing the same information, which is how the context demagnification changes as the distance from the focus increases.

There are a great number of geometric focus + context techniques that have been developed. Here, we present a review of the most fundamental and influential methods. A good survey and taxonomy of these distortion-based approaches is presented by Leung and Apperley [93].

The *Bifocal Display* [146] is the earliest and simplest geometric focus + context technique in information visualisation. It divides the screen into 9 regions, as shown in Figure 1.12. The central focus region has no demagnification, whereas the remaining regions have constant demagnification in the x direction, y direction or both, as appropriate. This gives the magnification function and 2 dimensional Cartesian application to a grid shown in Figure 1.13. One application of the Bifocal Display is the Table Lens [128], and a more esoteric application is the hardware Focus + Context

Figure 1.8: An Overview + Detail view of the city of San Francisco.

Figure 1.9: An Inverted Overview + Detail view of the city of San Francisco, such as used by the DragMag system [170].

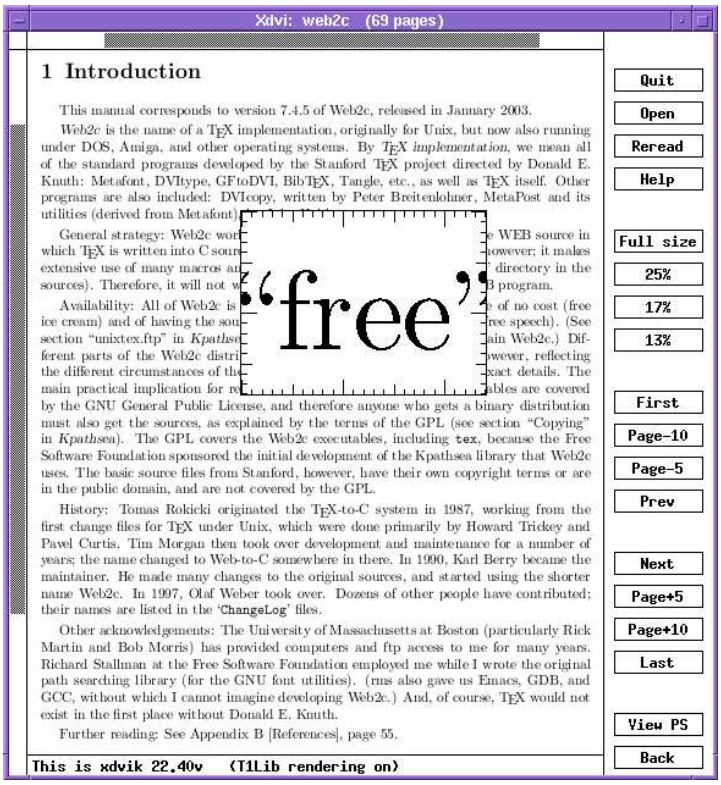


Figure 1.10: The `xdvi` program is an example of an Inverted Detail + Overview where the detail window is placed directly over the overview window, leading to a loss of immediate context.

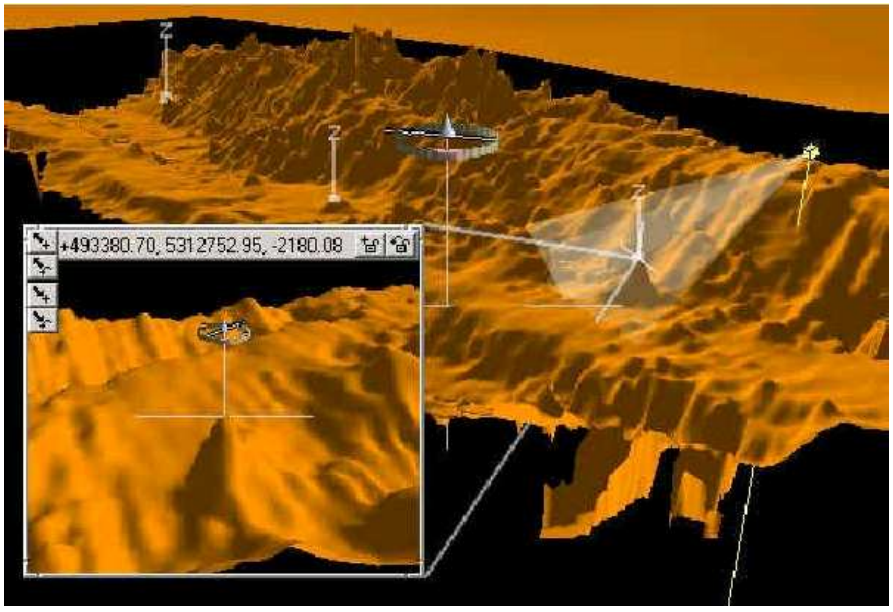


Figure 1.11: An example of tethered detail windows in GeoZUI3D. Courtesy of Matthew Plumlee and Colin Ware [120].

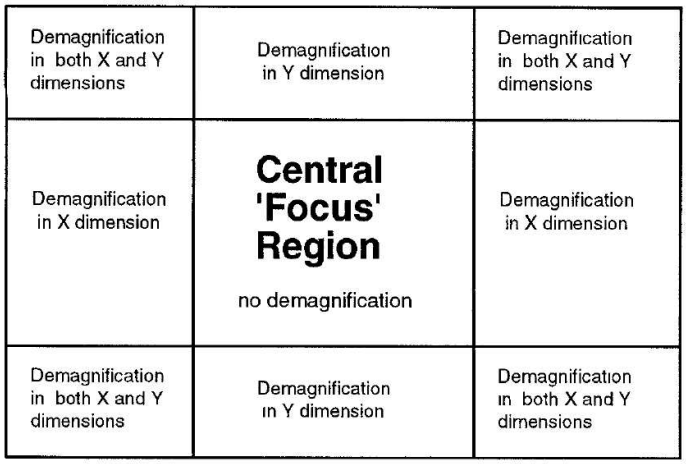
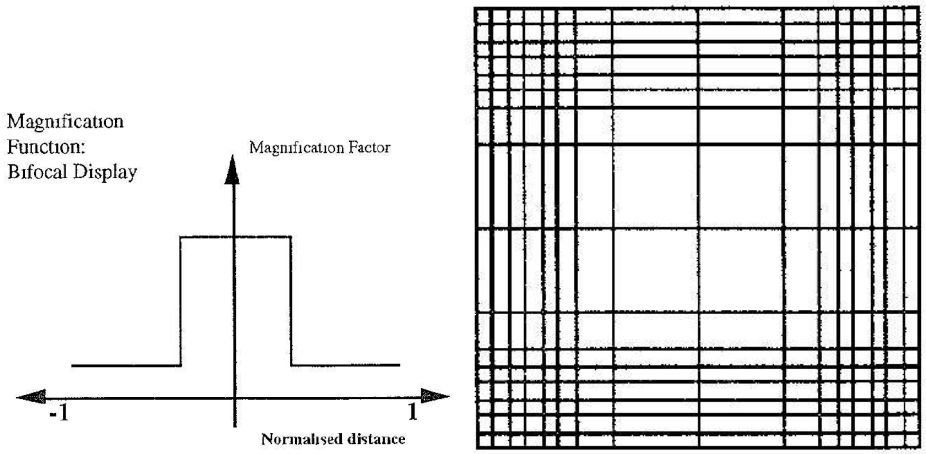


Figure 1.12: The 9 regions of a 2D Bifocal Display, from [93].

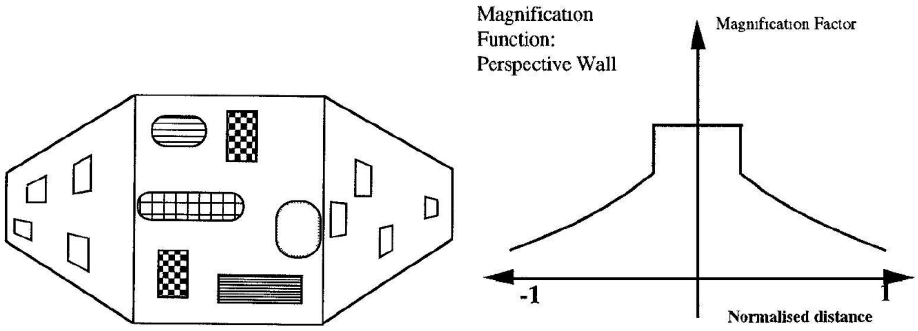


(a) Magnification function (b) 2D Cartesian

Figure 1.13: The magnification function and 2D Cartesian application of the Bifocal Display, from [93].

display device [18].

The *Perspective Wall* [95] works by "projecting" the visualisation onto a specifically configured "virtual wall". The main focus region consists of a section of the wall in the middle, that is parallel to the plane of the display device. To either side is the context region, consisting of the remainder of the wall receding into the distance at an angle. Figure 1.14 shows this approach, along with its magnification factor. The Perspective Wall adds a "3D feel" to the otherwise "flat" distortion techniques, but it can only apply the distortion in one dimension. The user accesses the context data by using the mouse to "scroll" the projection of the view left and right on the wall. The *Document*



(a) Visual appearance (b) Magnification function

Figure 1.14: The visual appearance and magnification function of the Perspective Wall, from [93].

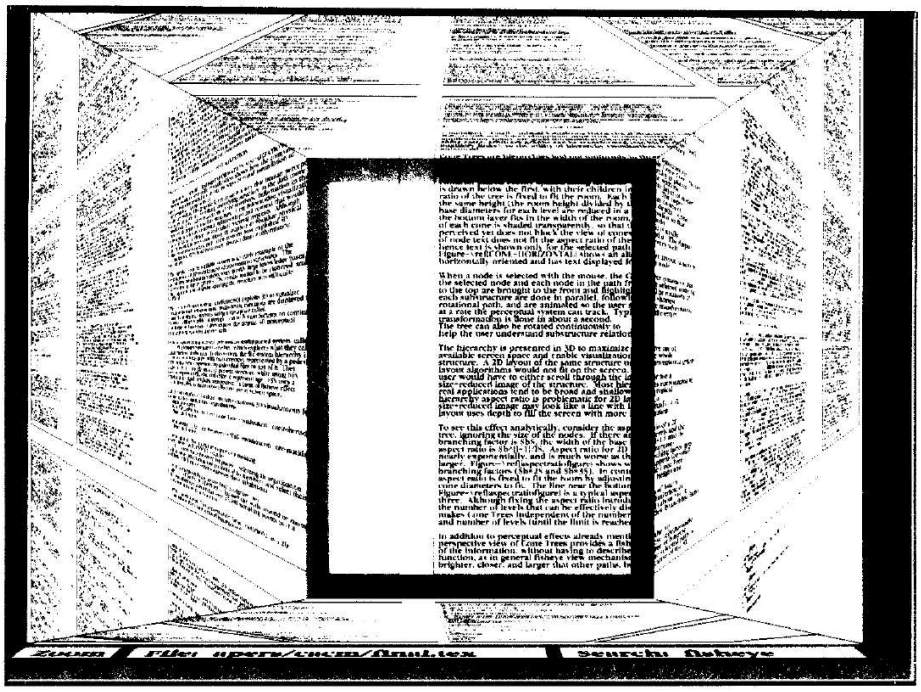
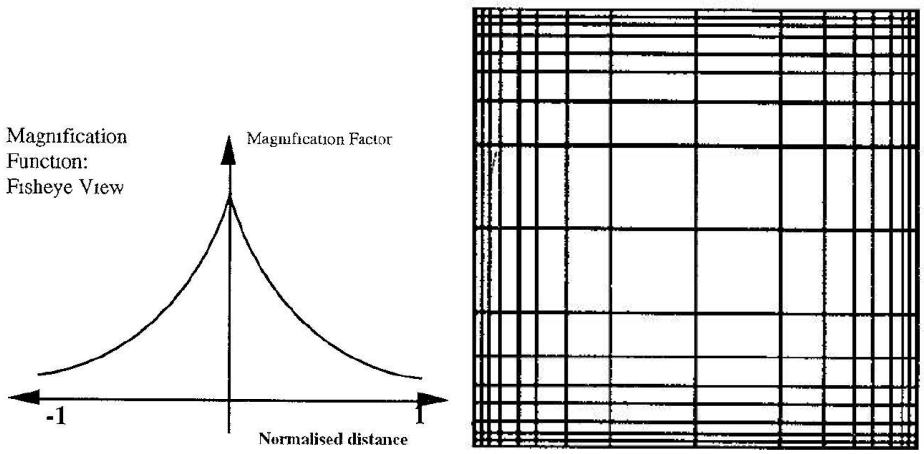


Figure 1.15: The Document Lens, from [130].



(a) Magnification function (b) 2D Cartesian

Figure 1.16: The magnification function and 2D Cartesian application of the Graphical Fisheye View, from [93].

Lens [130], shown in Figure 1.15, is a method of applying a perspective distortion effect in two dimensions.

The *Graphical Fisheye View* [136, 137] is a Focus + Context technique that has a smoothly varying magnification function, as shown in Figure 1.16. This means that there is no clear distinction between the focus and context regions, however, it is still true that the focus region has a higher magnification than the context region which surrounds it. An example of a tree drawn using the Graphical Fisheye View is shown in Figure 1.17. The central focus region is defined by the *focus point* which is always in the centre of the display. The user navigates through the data by clicking the desired position of the new focus point. This gives the user access to the entire dataset, even though, at any given time, the majority of the data is distorted. In addition, the transition between focus points can be animated by linearly interpolating the position of the focus point. The name "fisheye" comes by analogy to a fisheye lens in photography, which yields much the same distortion. It was originally derived from the Generalised Fisheye View, described below in Section 1.2.4. The *Hyperbolic Browser* [92] is an application of Graphical Fisheye Views to large trees; an example is shown in Figures 1.18 and 1.19. The Hyperbolic Browser have also been extended to 3D visualisation of large tree data [28, 104].

The *SHriMP visualisation system* [148] combines graphical fisheye views with the principles of mental map preservation and Semantic Zooming in the field of software visualisation. It also makes

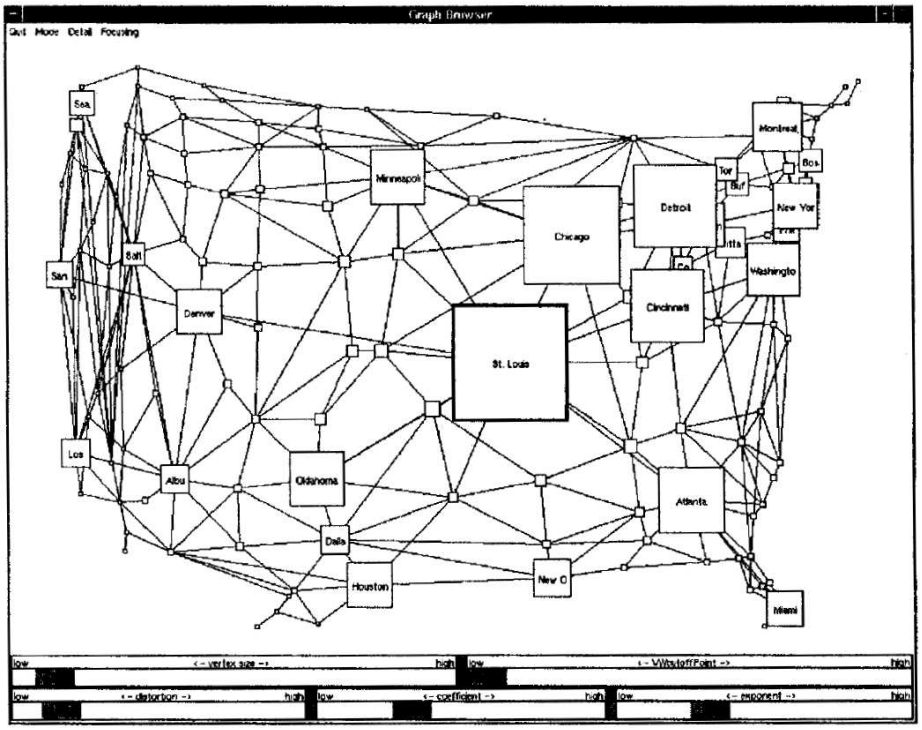


Figure 1.17: A Graphical Fisheye View of major highways in the USA [136, 137].

use of nested graphs in a similar way to that described below in Section 1.2.4.

Focus + Context techniques based on Graphical Fisheye Views have proved popular, being applied to a wide variety of problems, for instance, to long lists of menu items [20], and to Internet search engine results [113].

It is interesting to note that these techniques bear more than a passing similarity to some of the works of M. C. Escher. In particular, Graphical Fisheye Views use a similar geometric distortion to that in ‘Balcony’, shown in Figure 1.20, and also ‘Hand with Globe’, ‘Still and Reflection with Globe’ and ‘Three Spheres’. The Hyperbolic Browser was originally inspired by Escher’s ‘Circle Limit IV’, shown in Figure 1.21.

1.2.4 Non-geometric focus + context

It is possible to achieve Focus + Context by strategically hiding, or *eliding*, context data, rather than by geometrically distorting it.

The seminal work in Focus + Context is Furnas’s *Generalised Fisheye Views* and associated *Degree of Interest (DOI)* and *A-Priori Interest (API)* measures [58, 59]. Here, he was concerned

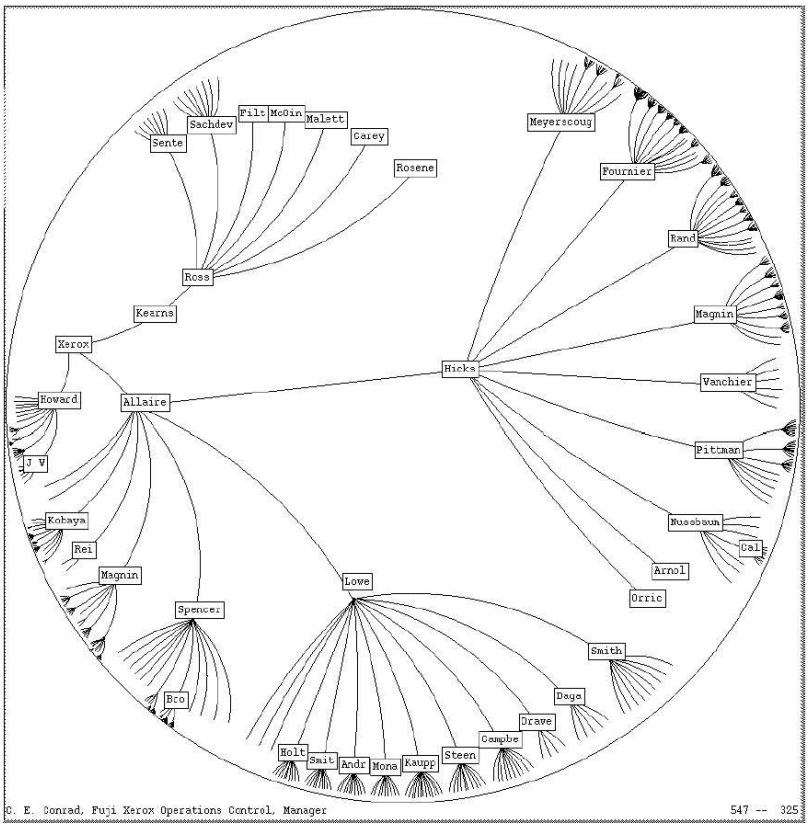


Figure 1.18: A Hyperbolic Browser view of an organisational chart [92].

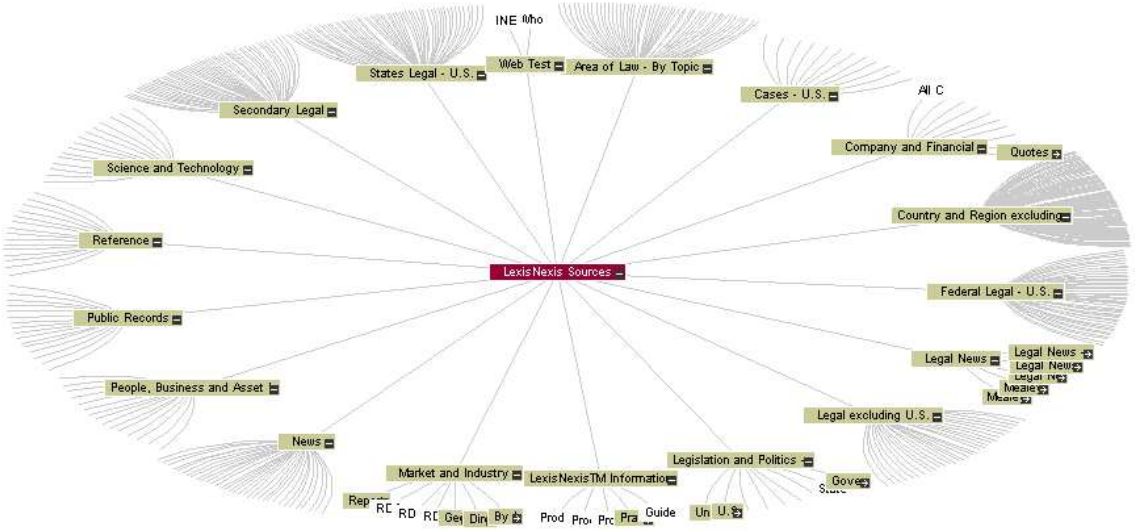


Figure 1.19: An Inxight StarTree [79] view of the LexisNexis online sources [94].

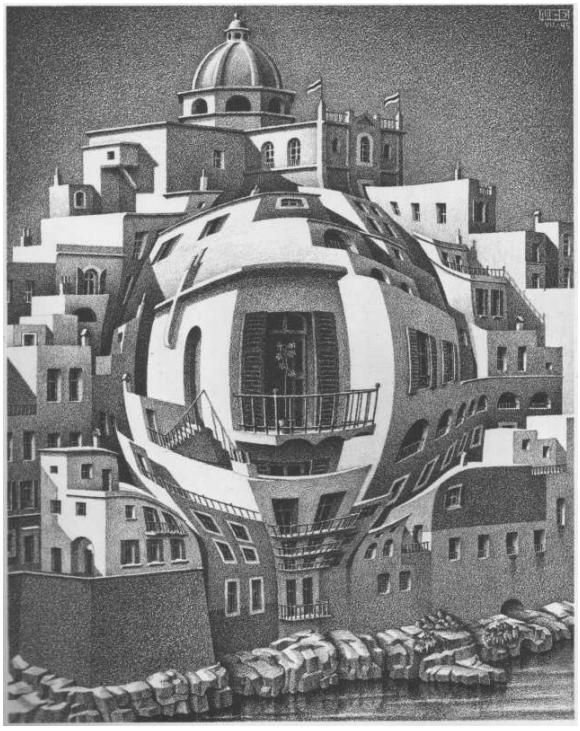


Figure 1.20: M.C. Escher's "Balcony" (c) 2004 The M.C. Escher Company - The Netherlands. All rights reserved. Used by permission.

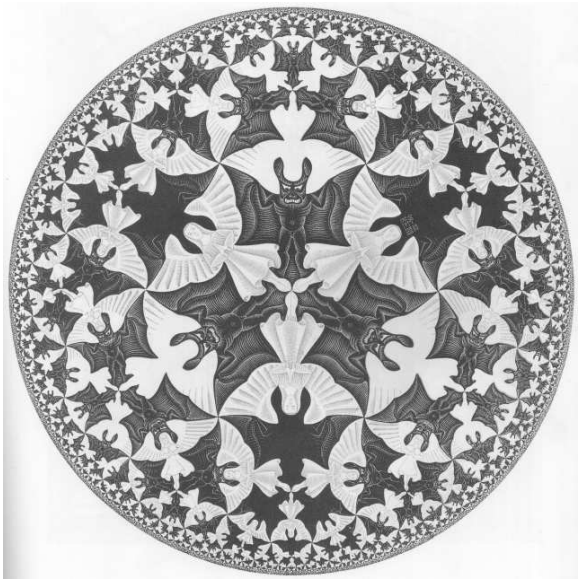


Figure 1.21: M.C. Escher's "Circle Limit IV" (c) 2004 The M.C. Escher Company - The Netherlands. All rights reserved. Used by permission.

```
   1 #define DIG 40
   2 #include <stdio.h>
...4 main()
   5 {
   6       int c, i, x[DIG/4], t[DIG/4], k = DIG/4, noprint = 0;
...8       while((c=getchar()) != EOF){
   9             if(c >= '0' && c <= '9'){
..16             } else {
  17                   switch(c){
  18                         case '+':
..27                       case '-':
..38                       case 'e':
>>39                             for(i=0;i<k;i++) t[i] = x[i];
  40                             break;
  41                       case 'q':
..43                       default:
..46                   }
  47                   if(!noprint){
..57                   }
  58             }
  59             noprint = 0;
  60       }
  61 }
```

Figure 1.22: An example of a Generalised Fisheye View on a section of C source code [59]. The left margin contains line numbers and elipses '...' for lines that are representative of hidden lines.

with textual listings of program source code, and elided the contents of control structures and functions that the user was not currently interested in, while still providing a visual indication of their existance. For example, an `if` statement may be shown with its condition, but not the code that executes when it is true. Figure 1.22 shows an example of this. Furnas's work was highly influential, and most Focus + Context systems use some of the ideas and concepts he presented.

Although the majority of research in Focus + Context techniques since Furnas's original presentation has emphasised its geometric or graphical nature, there are two techniques that combine non-geometric and geometric Focus + Context techniques. The first is Noik's "clustered graph fisheye view", and the second is the family of Variable, Continuous and Intelligent Zooming techniques.

Noik's *clustered graph fisheye view* [108, 109, 110, 111] applies Furnas's Degree of Interest to clustered graphs, using this as the basis of a graphical fisheye view. The result is a visualisation that resembles a geometrically distorting fisheye view, except that the focus and context magnifications are based on the graph theoretic distance between nodes, rather than the Euclidean distance. As such, there is no explicit magnification function: a notable distinction to geometric Focus + Context techniques. An example of this type of view is shown in Figure 1.23.

The *Variable Zooming* [138], *Continuous Zooming* [12, 14, 33] and *Intelligent Zooming* techniques [11, 13] introduce the idea of using the *cluster hierarchy* of a clustered graph as the main

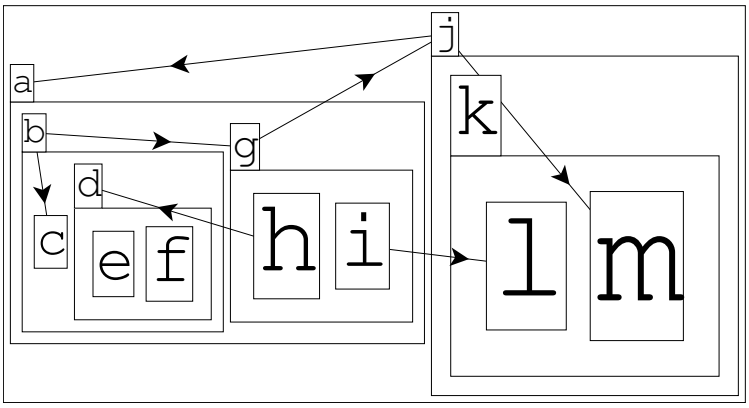
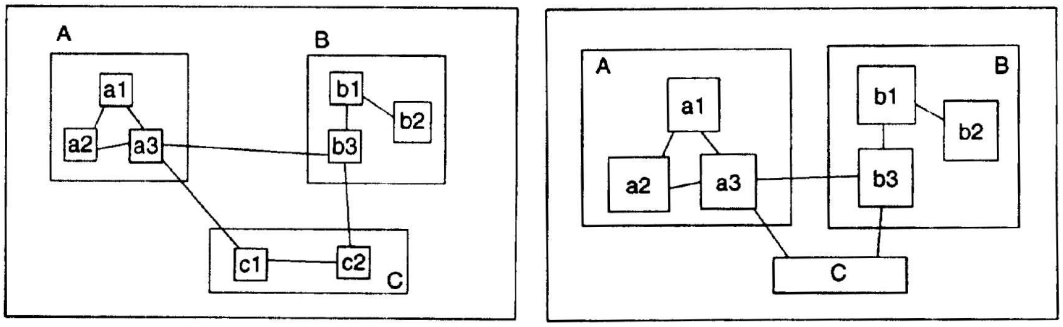


Figure 1.23: An example of Noik's clustered graph fisheye view [110].



(a) All clusters open

(b) Cluster C closed

Figure 1.24: An example of the Continuous Zoom system [12]. Notice that closing cluster C has the effect of hiding nodes c1 and c2, as well as increasing the sizes of nodes a1, a2, a3, b1, b2 and b3 since more space is available to them.

basis for controlling the level of detail in the Focus + Context system. The clustered graph is obtained by supplementing a graph with a hierarchical grouping, or *clustering*, of its nodes. These clusters may be *open*, to show the detail of the nodes within, or *closed*, to hide this detail. This work was applied to the domain of control system supervision, and the sizes of the nodes adjusted according to a Degree of Interest function similar to Noik's, in order to highlight the relative importance of nodes. In addition, when nodes became large enough to support it, relevent low-level detail of the control system sensors (such as time-series temperature data) is embedded within nodes, and thus also incorporates elements of Semantic Zooming. Figure 1.24 shows an example of the Continuous Zoom system.

Similar systems for viewing nested graphs in 3D (though without any further Focus + Context

considerations) have been developed [39, 114].

1.2.5 Animation

For the purposes of this thesis, an *animation* is a sequence of still images, known as *frames*, that give rise to the illusion of motion when displayed in rapid succession. While the difference between any two successive frames is small, it is structured such that the overall difference is seen over time as motion. This phenomenon is *apparent motion*, and is the basis of the frame-based motion exhibited in cinema, television and computer displays. The speed at which the frames are displayed is the *frame rate*, and is expressed in the number of *frames per second* (*fps*) or sometimes the time between frames (in milliseconds). Animations are symmetric in time, and the *time-reversed* version of an animation is easy to construct, simply by reversing the order in which the frames are displayed.

Animation can exhibit subtle problems not normally found in static visual presentations [166]. For example, the illusion of motion breaks down at about 10 fps, and 25 fps is generally agreed upon as a reasonable minimum frame rate. In addition, since frames may be a discrete sampling of an otherwise continuous motion, sampling effects known as *temporal aliasing* may occur. This is sometimes referred to as the *wagon wheel illusion*, in reference to the particular case where the wheel of a wagon appears to be spinning backwards when the wagon is travelling at certain speeds.

Motion has several beneficial uses in information visualisation. One of the most important is preserving the mental map during transitions between visualisations [21, 54, 55]. It has also been explored for highlighting the results of queries [15, 167] and aiding depth perception of 3D displays on 2D display devices [169]. A comprehensive taxonomy of motion-uses, in information visualisation, is given by Bartram in [16].

1.3 Aims

The Geometric Focus + Context techniques, particularly the Graphical Fisheye View and Hyperbolic Browser, are useful and common applications of the Focus + Context paradigm. They have the following benefits.

High Detail Focus: A small subset of the data is shown in the focus region with nearly no visual compression, allowing it to have high detail. This reflects the fact that the focus region shows the data the user is most interested in.

Complete Context: The remainder of the data is shown in the context region with a large degree of visual compression. This means that the normalised data content is 1 (even though the context region has low detail), providing complete context to the user.

Natural Metaphor: The metaphor employed by geometric Focus + Context techniques involves either lens- or perspective-based distortion functions, both of which are readily understood in the physical world.

Animated Navigation: The navigation operation, relocating the geometric focus point, is easily animated by presenting intermediate frames for interpolated focus points between the initial and final focus points.

Continuous Focus–Context Transition: The spatial transition between focus and context regions is continuous. This means that connections between data elements in the focus and context regions are preserved, supporting navigation. In some geometric Focus + Context techniques, such as the Fisheye View and Hyperbolic browser, the transition is also smooth (c.f. Bifocal Display).

The first two benefits, High Detail Focus and Complete Context, are the main advantages, and are the reason Geometric Focus + Context techniques are frequently used for the visualisation of large datasets. The main disadvantages of Geometric Focus + Context techniques are as follows.

Low Focus Data Density: Although the focus region is the area with the highest detail, and is the user's area of interest, it is frequently the case that the data content in this region is low. This problem is clearly visible in Figures 1.18 and 1.19.

High Context Data Density: As a result of visually compressing a large amount of data into the relatively small context region, this region often suffers from high density. This results in a crowded display with a high visual complexity. This is often compounded by aliasing effects, which are more pronounced due to the small number of pixels used per graphical object in the context region. Semantic zooming can help alleviate this problem by choosing low detail representations of sections of the context, or by eliding parts of it completely. This problem is clearly visible in Figures 1.17, 1.18 and 1.19.

Poor Spatial Properties: The introduction of a geometric distortion removes many of the properties which are present in undistorted visualisations. For example, relative orientations of data items may not be preserved, distance metrics depend on the location of the focus point, the transformation may not be affine, and area is never preserved. Figure 1.17 shows these problems; since this diagram shows a Cartesian Fisheye View, it is not as poor as diagrams that use radial Fisheye Views.

Difficult Multiple Foci: The presentation of more than one focus region is frequently difficult, and substantially more complicated, than the single focus case. This is covered in more detail in [93].

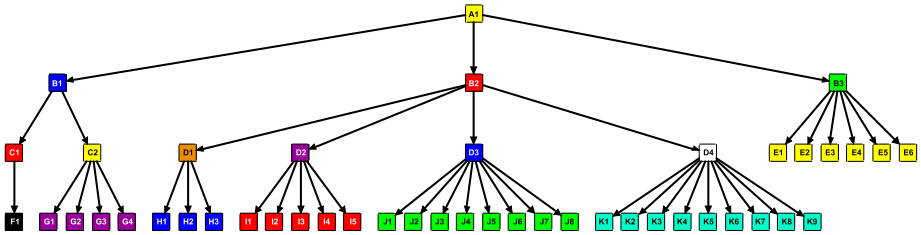
The motivation for this thesis is to determine a Focus + Context method that incorporates the best aspects of Geometric Focus + Context techniques while avoiding the problems. We introduce *Structural Zooming*, a Focus + Context technique with the central theme that displaying the data of the context region at lower detail (that is, at a higher level of abstraction) is preferable to geometrically distorting all of the context data. This can be thought of as "abstracting" or "summarising" the context, allowing the user to see a brief summary of its contents and its relation to the focus region.

The solution presented by Structural Zooming must address all the problems with Geometric Focus + Context, without compromising any of its advantages. Thus, we aim for a solution that has these advantages over Geometric Focus + Context:

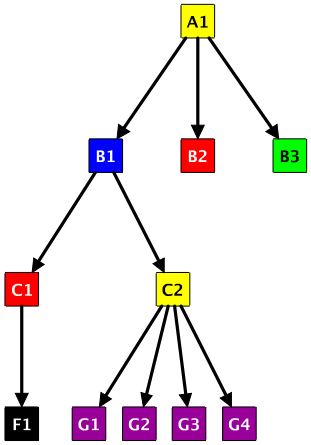
Mixed Detail View: The data is shown at a combination of high and low detail, as appropriate. This supports a Focus + Context type of visualisation.

Uniform Magnification: All data is displayed at the same magnification level: this makes it easier to read data in the context region and compare it with that in the focus region.

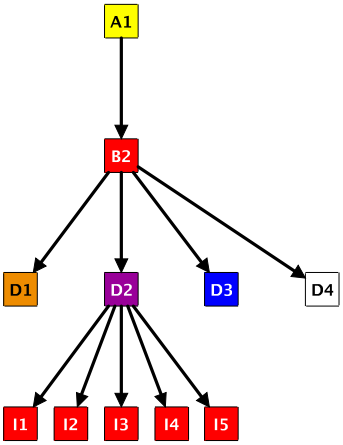
Constant Visual Complexity: The complexity of the view presented is kept approximately constant. This helps to overcome the problems of Low Focus Data Density and High Context Data Density. "Visual complexity" is explained in more detail in Chapter 3, but an illustrative example is shown in Figure 1.25.



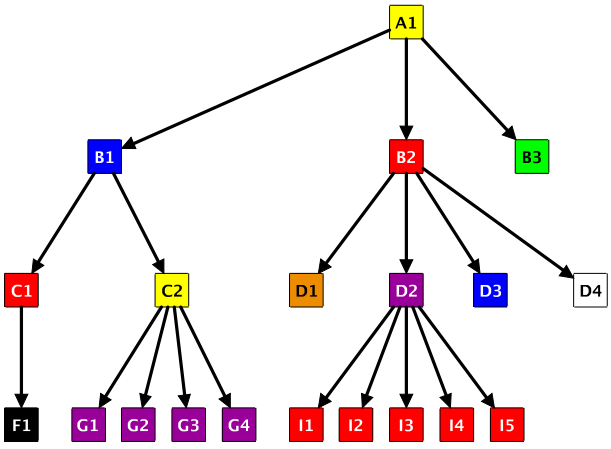
(a) Full tree



(b) Focusing on node B1



(c) Focusing on node D2



(d) Focusing on nodes B1 and D2

Figure 1.25: An example of the meaning of "constant visual complexity". When "focusing" on a node, it is shown with its descendant nodes, siblings and ancestors. (a) shows the full tree. (b) shows the view where the user has focused onto node B1; 11 nodes are visible. If the user then chooses to shift the focus to node D2, the view may shift to that shown in (c). In this case, the node B1 and its descendants have been hidden so that there may still be only 11 nodes visible, and so the visual complexity has been preserved (to some approximation, at least). This is in contrast to (d), where the focus is with node D2 but without removing the previously focused node B1. This case has 20 nodes, which is approximately twice as many as (b) and (c).

Implicit Focus and Context Regions: The focus and context regions are not specified explicitly as they are in geometric zooming. The focus-region is simply that occupied by the focus data, and similarly for the context region. This allows these regions to move and rearrange as necessary.

Leverages Existing Techniques: Existing standard algorithms and techniques are used for data layout, with their benefits being applied to the mixed detail focus + context data. For example, possible algorithms include graph drawing algorithms and space-filling techniques such as treemaps. This ensures good aesthetics and high quality layout, which sometimes cannot be guaranteed in geometric distortion.

Uniform Spatial Density: In many situations, uniform magnification, implicit focus and context regions, and the use of existing techniques mean that the geometric zooming problems of low focus density and high context density are overcome.

Structural Zooming is most closely related to Continuous and Intelligent Zooming, which are in turn related to Generalised Fisheye Views and Semantic Zooming. Structural Zooming supports multiple levels of detail (such as by opening and closing clusters) in a similar way to Continuous and Intelligent Zooming, but does not geometrically distort the sizes of the nodes. Like Generalised Fisheyes, it does not use geometrically-based magnification, but unlike them it does not require explicit DOI or API measures. It uses distinct high and low detail representations of data in the same way as Semantic Zooming, but in addition it allows different spatial regions of the display to be seamlessly shown at different levels of detail. It is also driven by the user's logical navigation, rather than by a simplistic magnification factor as Semantic Zooming is.

1.4 Research methodology

As the detail-context tradeoff is a fundamental problem in visualisation, its solutions tend to be applicable to many different types of visualisations. The research methodology followed is primarily *empirical* in nature. It consists of three broad stages, *design*, *implementation* and *validation*. The Structural Zooming technique is designed in two distinct levels. The first addresses the problem in a general, visualisation-independant setting. This presentation allows a visualisation designer to

adapt the Structural Zooming technique to any particular type of visualisation. This general presentation is then used to design a method of Structural Zooming for relational data, specifically, trees and clustered graphs. This method for the Structural Zooming of relational data is implemented in a system which allows the method to be experimentally validated and evaluated. This validation is performed in an application-based context, using datasets from three particular domains. The criteria for evaluation is a collection of *quality measures*, each of which quantitatively captures the essence of a desirable visualisation property. This allows the performance of Structural Zooming to be observed as input parameters are changed individually.

1.5 Contributions

This section briefly describes the main contributions of this thesis and where each may be found.

- An empirical investigation into the definition of the "size" of a rectangle that is most useful for inclusion tree diagrams is presented in Section 2.3.
- A framework for determining the visual complexity of a visualisation is presented in Section 3.1.
- The technique of Structural Zooming is defined and presented in a general, visualisation-independant way in Chapter 3.
- Structural Zooming is applied to the Focus + Context visualisation of relational data in Chapter 4. Methods for two types of relational data are presented, trees and clustered graphs. Structural Zooming of trees is presented in the traditional node–link style and in inclusion style.
- A "stable" version of the Jewellery Box Inclusion Layout Algorithm is presented in Section 4.2.1. This attempts to minimise changes and preserve the orthogonal ordering of arbitrary inclusion tree layouts.
- A method for the animation of orthogonal edge layouts is presented in Section 4.3.
- Measures for determining the quality of animated transitions in Structural Zooming of trees and clustered graphs are presented in Section 4.4.

- Methods for automating the generation of "navigation paths" in Structural Zooming of trees and clustered graphs are presented in Section 4.5. Navigation paths are use-cases for Structural Zooming, and are important for evaluation.

- An empirical investigation into Structural Zooming of relational data is presented, using data from the three application areas of Design Behaviour Trees (Chapter 5), Software Views (Chapter 6) and Citation Networks (Chapter 7). The results of this investigation are discussed and interpreted in Chapter 8.

1.6 Organisation of the thesis

Chapter 2 presents background aspects of the relevant areas of research, providing the pertinent terms, definitions, concepts and results that are required for the remainder of the thesis. Chapter 3 presents the concepts and model for the general technique of Structural Zooming of arbitrary visualisations. Chapter 4 shows the application of Structural Zooming to relational information — in particular, trees and clustered graphs. Chapter 5 presents the first application of Structural Zooming to a concrete type of data, that of Design Behaviour Trees. Chapter 6 presents the second application, that of views of software systems. Chapter 7 presents the third and final application, that of citation networks. Chapter 8 discusses and interprets the results from the three preceeding chapters. Finally, Chapter 9 sets out the thesis' concluding remarks. Appendix A contains a description of the contents of the CD-ROM accompanying this thesis.

A large part of this thesis is concerned with animations, and as such, it makes use of *animated figures* to show illustrative animations that support the text. Since it is not possible to show motion on paper, these figures appear in the printed form of this thesis as a matrix of frames. Additionally, the electronic PDF version embeds the animated figures using the *animfig* package [122]. This allows interactive playback of the animations in PDF viewers that support Acrobat PDF Javascript, such as the freely available Adobe Acrobat Reader, using a method based on that described in [72]. These embedded animations are accessed by clicking on the static matrix of frames. The electronic version is provided on the CD-ROM accompanying this thesis, described in Appendix A, and on the World Wide Web at `http://www.kev.pulo.com.au/thesis/`.

Portions of the material presented in this thesis have been published in [123] and [124]. The author of this thesis was the primary author of both papers, with the co-authors providing direct supervision.

CHAPTER 2

Background

This chapter introduces background material for the types of data on which this thesis focusses, namely, trees and clustered graphs. Section 2.1 introduces basic concepts and definitions, while Section 2.2 reviews methods for automatically drawing trees and clustered graphs. Section 2.3 presents the results of an empirical investigation into a specific design parameter for the layout of certain types of trees.

2.1 Basic concepts

This section describes the main relational data structures considered in this thesis, that is, trees and clustered graphs. Methods of visual presentation are given, and the relationship between trees and clustered graphs is explained.

2.1.1 Trees

'Trees' are fundamental data structures that are ubiquitous throughout all areas of computer science and information systems, as well as many of the physical and empirical sciences. For example, in the field of operating systems there are process-trees [83] and filesystem B-trees [29], in databases there are query-trees [163], in languages there are parse and expression-trees [4], and in geographic information systems (GIS) there are various spatial data structures such as quadtrees, PR-trees and k-d trees [132, 133].

A *tree* (more correctly, a *rooted tree*) is defined recursively as a node v and a collection of *child* trees, that is,

$$T = (v, \{T_1, T_2, ..., T_{d_v}\})$$

where T_i are similarly defined trees. The set of all nodes is denoted by V. The nodes v_1, v_2, ..., v_m

are the *children* of v, and v is their *parent*. Two nodes u and v with the same parent are *siblings*. The set of children of a node v is denoted by $C(v)$, and the number of children $d_v = |C(v)|$ is the *degree* or *branching factor* of v. Nodes that have $d_v = 0$ (that is, no children) are *leaf* nodes, and nodes which have $d_v > 0$ are *non-leaf* or *internal* nodes. The node of T which is not a child of any other node is the *root node*, denoted as r. A *binary tree* is one in which the degree of every internal node is at most 2.

The relationship between a parent node u and its child v is the *edge* (u, v). The set of all edges in the tree is denoted by E. A *path* is a list of nodes $(v_1, v_2, ..., v_m)$ such that $(v_i, v_{i+1}) \in E \ \forall\, 1 \leq i \leq m-1$. A node u is an *ancestor* of v if it is possible to reach u from v by traversing only parent edges, and v is then a *descendant* of u. The set of ancestors of a node v is denoted by $A(v)$, and the set of descendants by $D(v)$.

There are two broad styles for the visual representation of trees. Traditionally, trees have been drawn using the *node-link* style, which is still the most common style in use today. In this style, tree nodes are drawn as closed connected regions in the plane, and edges are drawn as lines (with or without arrow heads) connecting the node regions. Several algorithms have been devised for drawing trees in the node-link style, including the famous Reingold-Tilford [129] algorithm which arranges nodes on horizontal layers. More formally, a node-link layout of a tree T defines a region R_u in the plane $\mathbb{R}^2$ for each node u and a curve $L(u, v)$ for each edge (u, v), such that:

- No two node regions overlap, that is, $R_v \cap R_w = \emptyset \ \forall\, v, w$.
- For all edges $(u, v) \in E$, R_u is connected to R_v by the curve $L(u, v)$.

The curve $L(u, v)$ is simply a line, which may be curved or straight and may consist of one or more connected segments. These curves may or may not have arrowheads; if so, the arrow is directed toward the child node. These curves are described in more detail in Section 2.1.2.

The *inclusion* layout style [44] is an alternate method of drawing trees, where the parent–child relationship is visually represented by the closed geometric region of the child node being completely contained within the region of the parent node. More formally, an inclusion layout for a tree T defines a region R_u in the plane $\mathbb{R}^2$ for each node u of T, such that:

- If $w \in C(u)$ then $R_w \subset R_u$.
- If $v \in C(u)$ and $w \in C(u)$, then $R_v \cap R_w = \emptyset$ (that is, siblings do not overlap).

- If $v \in C(u)$ and $w \in C(u)$, then $d(p,q) \geq \delta \; \forall \; p \in R_v$ and $q \in R_w$ where $d(p,q)$ is the Euclidean distance metric between points p and q (that is, siblings are separated by a minimum distance of δ).

Non-leaf nodes are usually drawn in a manner that is visually distinct from leaf nodes, known as *containers*, which emphasises their role of grouping sibling nodes.

The overall size of a layout is determined by the size of the *bounding box*, which is the smallest isothetic rectangle[1] $\mathcal{R}$ such that $R_u \subset \mathcal{R} \; \forall \; u \in T$ and $L(u,v) \subset \mathcal{R} \; \forall \; (u,v) \in E$. Although the layout itself may be any general region, we take the bounding box to reflect the fact that ultimately, the layout must be presented on a standard rectangular computer display or sheet of paper. The region R_u for each node may be freely chosen, and is usually done in order to accommodate the display of the *label* of the node, which may, for example, be text, an image or a collection of shapes. The overall size of an inclusion layout is simply by the bounding box around R_r. This study is interested in tree drawings that minimise the size of the overall layout while preserving some aesthetic criteria, such as no overlaps between nodes. Minimising the size of the layout ensures that the amount of detail shown on-screen is maximised.

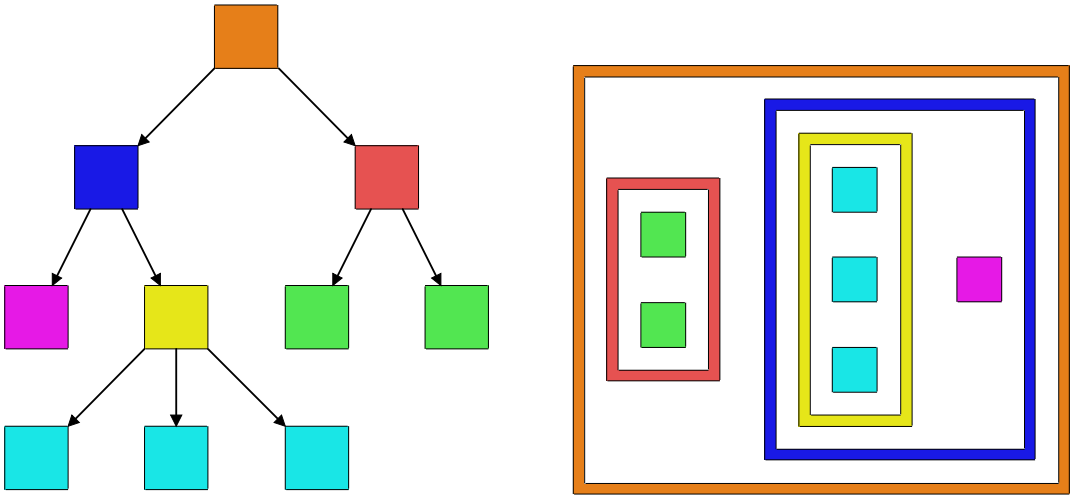
In most practical systems, node regions are only permitted to be isothetic rectangles. This is because most node labels are text or images — both of which are best represented within rectangles. In the situations where this is not the case, it is still possible to use the bounding box of the actual node label. In addition, the use of rectangles considerably simplifies the analysis of the problems and algorithms. For these reasons, this thesis deals only with the case of node regions that are isothetic rectangles.

Figure 2.1 illustrates an example tree in both the node-link and inclusion styles.

An *antichain* of a tree[2] is a set of nodes N such that no ancestor–descendant relationship exists between any pair of nodes $u, v \in N$. A *maximal antichain* is one in which the addition of any other node to N would cause it to no longer be an antichain. It is easy to see that any antichain can be extended to be a maximal antichain, thus, hereinafter, antichains are assumed to be maximal, unless otherwise noted. A *branch* is a path from the root node to a leaf node: clearly, there is exactly one branch for each leaf node. It is also easy to see that an antichain intersects every branch once (that is, exactly one node in every branch is in N) [5]. As such, an antichain may be considered to be a

[1] An *isothetic* or *orthogonally-aligned* rectangle is one with sides parallel to the x and y axes.

[2] Antichains are expressed more formally by representing the tree as a *partially ordered set*, or *poset*. However, such a treatment is unnecessarily general and this thesis considers only antichains of trees. Refer to Aigner [5] for a complete combinatorial treatment.



(a) Node-link Layout Style

(b) Inclusion Layout Style

Figure 2.1: An example tree in node-link and inclusion layout styles.

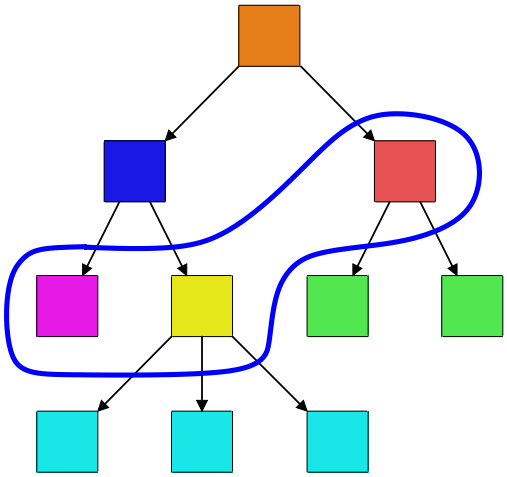


Figure 2.2: An example antichain of the tree shown in Figure 2.1.

"horizontal cut" through a top-down node-link drawing of a tree, as illustrated in Figure 2.2.

An *abridgement* of a tree is the visualisation of an antichain. The antichain N partitions the tree into two groups,

$$\begin{aligned} V_s &= \{v \mid v \in N \text{ or } v \in A(u) \text{ for some } u \in N\} \\ V_h &= \{v \mid v \in D(u) \text{ for some } u \in N\}. \end{aligned}$$

Thus, $r \in V_s$, $V_s \cap V_h = \emptyset$ and $V_s \cup V_h = V_T$. The set V_s is the set of *shown* nodes, and V_h is the set of *hidden* nodes. When displaying the abridgement, only nodes in V_s are drawn, and the nodes in V_h are said to have been *hidden* or *elided*. The non-leaf nodes of the antichain are *induced leaf nodes* of the abridgement, more simply known as *collapsed nodes*. Non-leaf nodes in V_s but not in N are *expanded nodes*.

Abridgements may be shown in either node-link or inclusion style. However, in the inclusion style, collapsed nodes are not drawn as containers, rather, they are drawn with the same visual representation as leaf nodes. In order to distinguish collapsed nodes, we use the convention of adding a visual cue to indicate that the collapsed node is a "representative" of its hidden descendants. Figure 2.3 shows the abridgement associated with the antichain from Figure 2.2, in both node-link and inclusion styles.

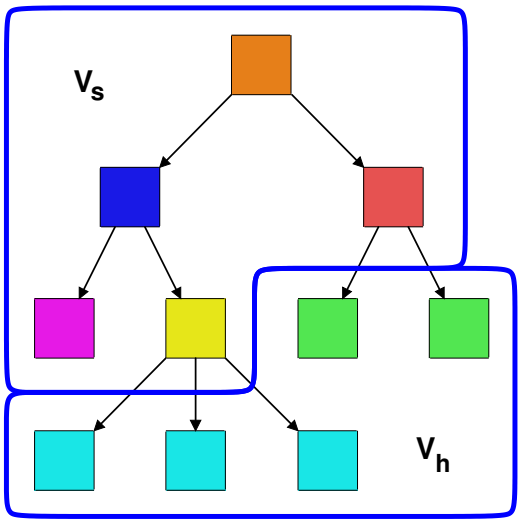
Another way to look at abridgements is through the use of logical *open* and *close* operations on nodes. Closing a node corresponds to moving its descendants from V_s to V_h, and opening a collapsed node corresponds to moving its immediate children from V_h to V_s. As discussed in Chapter 4, these operations form the basis for the navigation of relational data presented in this thesis.

This thesis is primarily concerned with the inclusion layout style, due to the strong relationship between inclusion layout and clustered graphs, which are discussed in Section 2.1.2. However, some consideration is given to a particular type of node-link layout in Section 2.2.1.

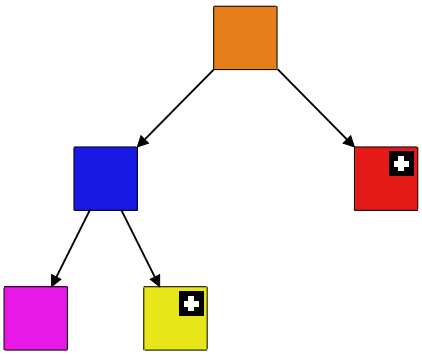
Whereas the size of any node in the node-link style is free, the size of a non-leaf node in the inclusion style depends primarily on the sizes of its children and how their positions are arranged. The rectangle for a non-leaf node in the inclusion style is generally given by the bounding box of its children, with some additional space for the node label.

The fundamental problem for inclusion layout is as follows:

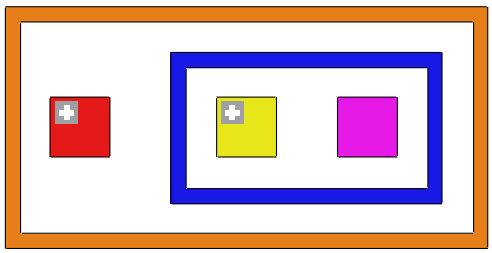
Minimum Inclusion Layout Problem (MILP): Given a tree T and a width X_v and



(a) Abridgement



(b) Node-link



(c) Inclusion

Figure 2.3: The abridgement associated with the antichain from Figure 2.2. The small "+" glyph in the corner of a node indicates it is collapsed.

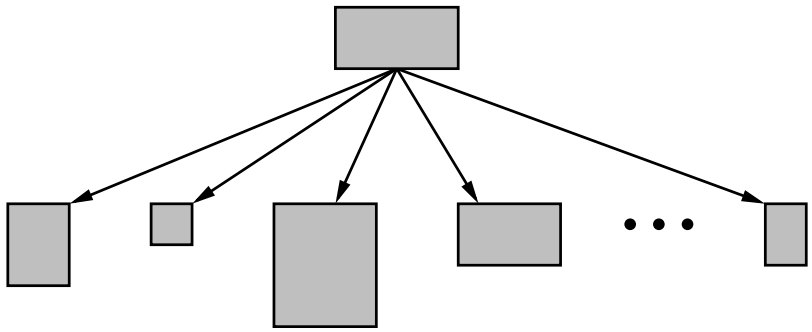


Figure 2.4: A tree which shows that MILP is reducible to the 2 dimensional bin packing problem.

height Y_v for each leaf v of T, find a minimum size inclusion layout for T such that for each leaf v, the dimensions of R_v are $X_v \times Y_v$.

However, there are several possible ways to define the "size" of a rectangle, and hence the size of a tree layout. For a rectangle of width x and height y, some possible *size measures* are:

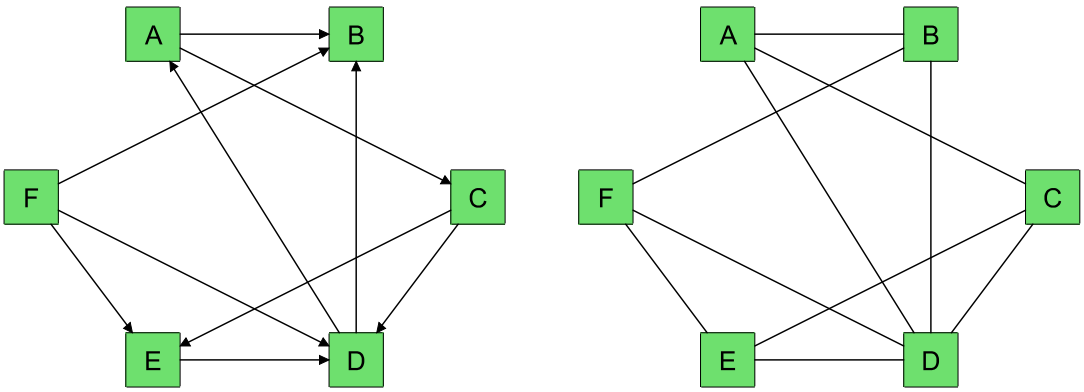
- area: $\psi(x, y) = xy$
- perimeter: $\psi(x, y) = x + y$
- minimum enclosing square: $\psi(x, y) = \max(x, y)$
- square aspect ratio: $\psi(x, y) = \left|\frac{x}{y} - 1\right|$

Note that the minimum enclosing square and square aspect ratio measures can easily be generalised to an arbitrary aspect ratio r, which is useful for non-square output regions (for example, many desktop computer displays have $r = 4/3$).

- minimum enclosing rectangle: $\psi(x, y, r) = \max(x,\, ry)$
- aspect ratio: $\psi(x, y, r) = \left|\frac{x}{y} - r\right|$

For simplicity, this study only considers the square ($r = 1$) cases. Section 2.3 shows that for the purposes of the argument, the minimum enclosing square size measure is the most appropriate.

If the tree in which every non-root node is a leaf, shown in Figure 2.4, is considered, it can be seen that MILP is equivalent to the 2 dimensional bin-packing problem, and is thus NP-hard [96]. Approaches for solving MILP are examined in Sections 2.2.1 and 2.2.2.



(a) Directed (b) Undirected

Figure 2.5: An example directed and undirected graph.

2.1.2 Clustered graphs

Graphs

Graphs, like trees, are commonplace in a wide variety of areas within computer science and many other applications and domains. Graphs are useful for modelling *relational data*, that is, data that involves *entities* or *objects*, and binary *relations* between them. For example, graphs — and the automatic visualisation of graphs, known as *graph drawing* [32] — are useful for metabolic pathways in the field of bioinformatics [98], circuit schematics in VLSI [10], visibility graphs for path planning in robotics [142], network flow graphs in transportation routing problems [29], neural and belief networks in artificial intelligence [131], fund manager flow graphs in finance [38], as well as a multitude of applications in software engineering [67, 73, 84, 102].

A *graph* $G = (V, E)$ is a set of *nodes* (or *vertices*) V and a set of *edges* E, where an edge is a pair of nodes, that is, $E \subseteq V^2$. A graph may be *directed*, in which case an edge is an ordered pair of nodes, or a graph may be *undirected*, in which case the pair of nodes is unordered. Graphs are almost always visually represented as node-link diagrams. Naturally, the edges in directed graphs have arrowheads, while those in undirected graphs do not. Figure 2.5 shows an example of a directed and an undirected graph.

Two nodes v_1 and v_2 are *connected* or *adjacent* if $\exists e \in E$ such that $e = (v_1, v_2)$, that is, $(v_1, v_2) \in E$. The edge $e = (v_1, v_2)$ is *incident* to nodes v_1 and v_2, and in a directed graph, v_1 is called the *source* node of e, and v_2 is the *target* node. The *degree* of a node is the number of edges

incident to it. For directed graphs the *out-degree* of a node is the number of edges for which it is a source node, and the *in-degree* is the number of edges for which it is a target node. A *path* is a list of nodes $(v_1, v_2, ..., v_m)$ such that $(v_i, v_{i+1}) \in E \ \ \forall\ 1 \leq i \leq m-1$. A *cycle* is a path $(v_1, ..., v_m)$ such that $(v_m, v_1) \in E$. An *undirected cycle* in a directed graph is defined similarly, except that it ignores the direction of the edges, that is, it only requires

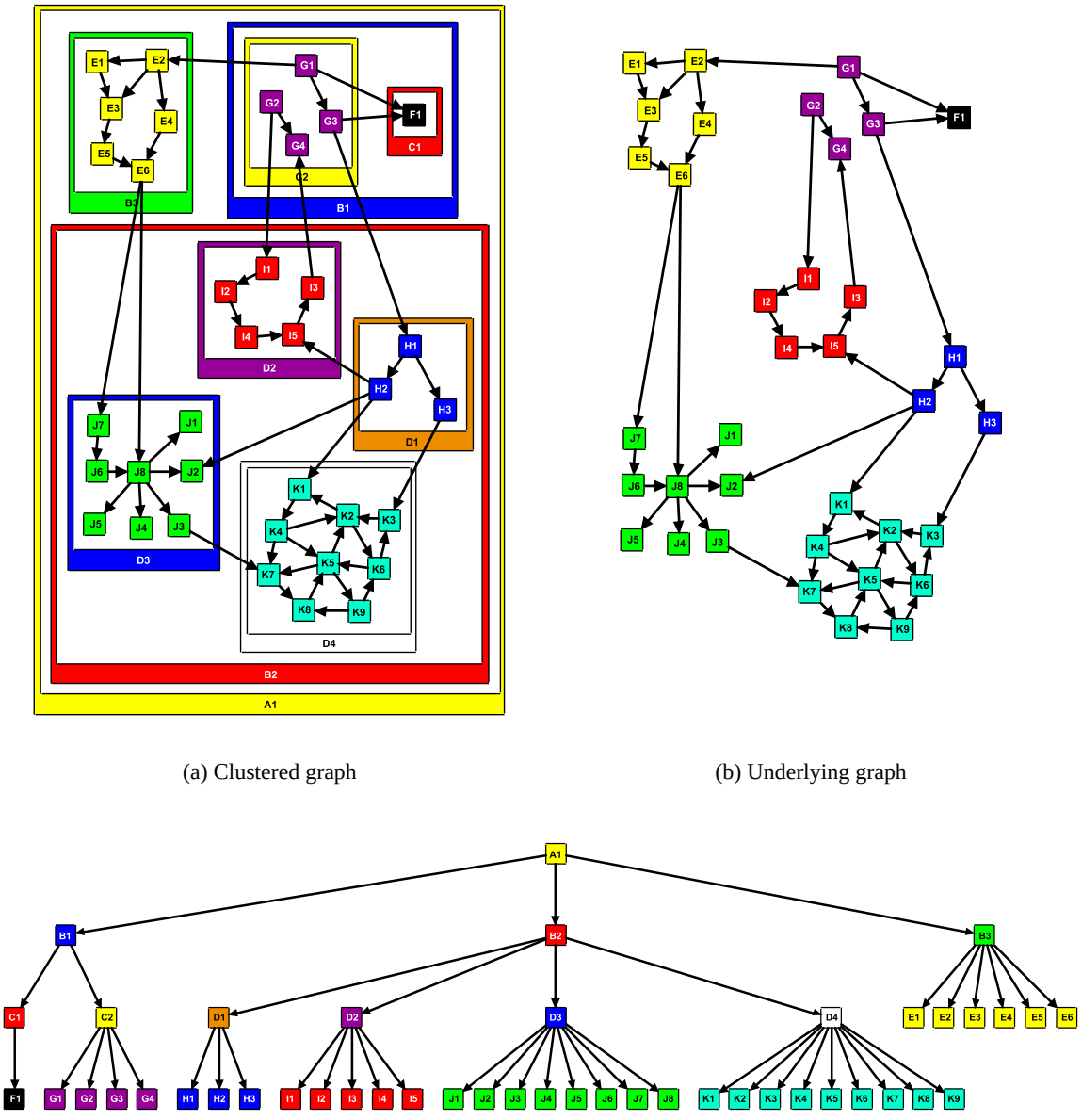
- Adjacency between v_i and v_{i+1}, that is, $(v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E$.
- Closure of the cycle, that is, $(v_m, v_1) \in E$ or $(v_1, v_m) \in E$.

A graph is *connected* if a path exists between all pairs of nodes, and *disconnected* otherwise.

A tree may be defined as an undirected connected graph which contains no cycles, or as a directed connected graph which contains no undirected cycles and an in-degree of each node of at most 1. This definition also permits *unrooted trees*, also known as *infinite trees*, which have no distinguished root node. This definition implies that a rooted tree with n nodes must have exactly $n-1$ edges, and has the corollary that a graph with n nodes and more than $n-1$ edges cannot be a root tree [172].

Clustered graphs

A *clustered graph* $C = (G(V_G, E_G), H(V_H, E_H))$ is an *underlying graph* G with an additional *cluster hierarchy* tree H. The nodes of the underlying graph are the leaf nodes of the cluster hierarchy, and so $V_G \subset V_H$. The remaining nodes are called *cluster nodes*. The edge sets E_G and E_H are disjoint, that is, $E_G \cap E_H = \emptyset$, since E_G contains only edges between underlying graph nodes, and E_H contains only edges between cluster nodes or edges between cluster nodes and graph nodes. The cluster hierarchy defines a recursive grouping, or *clustering*, of the nodes of G — hence the name "clustered graph". Clustered graphs are drawn in a similar fashion to normal graphs, but with the restriction that the nodes of H must be drawn using the inclusion layout style, that is, child graph and cluster nodes must be drawn within their parent. As with the inclusion layout style, cluster nodes are drawn as containers to emphasise their role of grouping cluster and graph nodes. Figure 2.6 shows an example clustered graph with its corresponding underlying graph and cluster hierarchy. This definition of clustered graph layout differs from the model presented in [47] which concentrates on the concept of *c-planarity*: a condition where edges are additionally not allowed to intersect clusters with which they have no relationship. *Compound graphs* [135, 151] and *higraphs* [70] are both more general graph models than the clustered graph model presented here; among



(a) Clustered graph

(b) Underlying graph

(c) Cluster hierarchy

Figure 2.6: An example of a clustered graph, also showing the underlying graph and cluster hierarchy. Cluster nodes begin with A–D, while graph nodes begin with E–K.

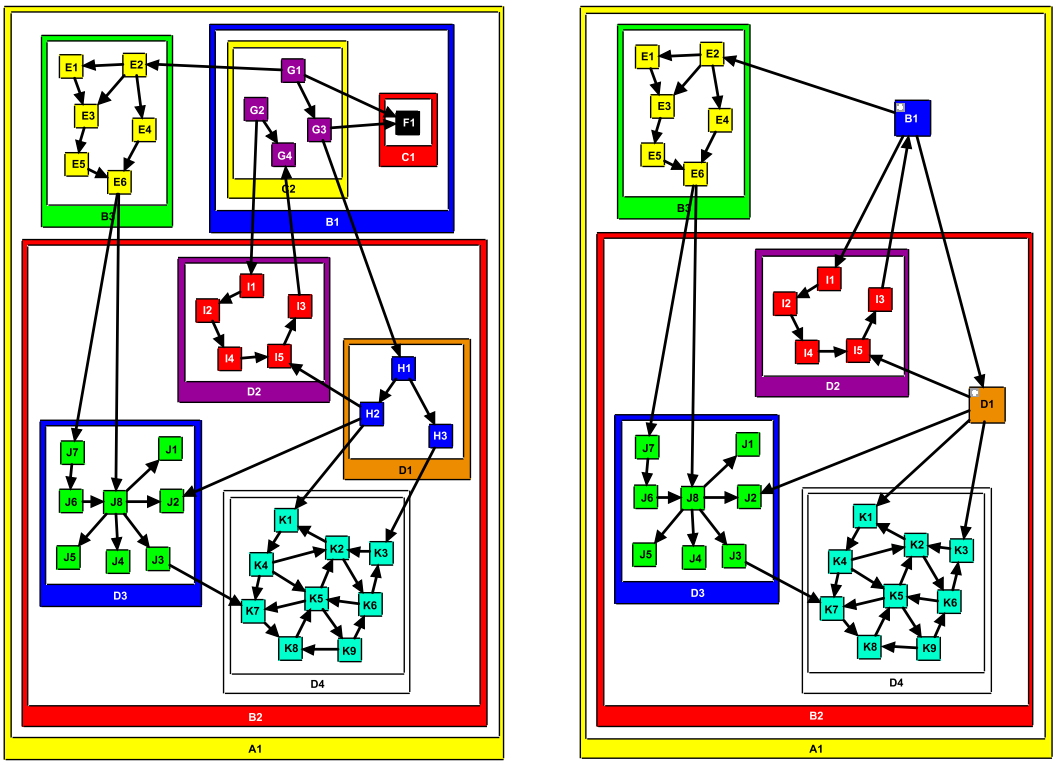
other differences, nodes are allowed to be members of more than one cluster and edges may join more than two nodes.

Clustered graphs may be considered to be a generalisation of normal graphs because they reduce to normal graphs when all the graph nodes are placed within a single cluster node. This produces a "flat" cluster hierarchy, in which every non-root node is a leaf, as shown previously in Figure 2.4.

Inclusion trees may be deemed to be clustered graphs where $E_G = \emptyset$. Alternatively, clustered graphs may be regarded as inclusion trees with additional node-link style edges between the leaves. This is the methodology used in this thesis — to initially approach the case for inclusion style trees, followed by the extension to clustered graphs by considering node-link edges in addition to the inclusion tree.

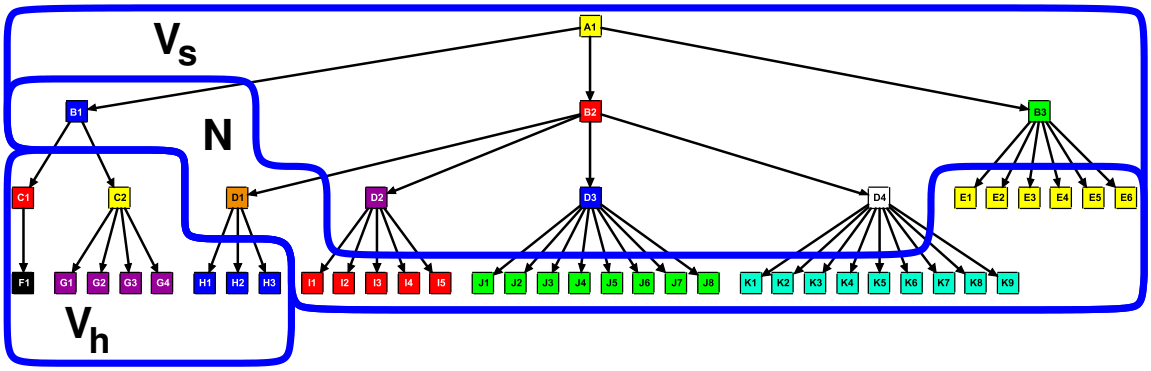
Abridgements may also be used for clustered graphs. In this case, there is the issue of how to deal with edges incident to hidden nodes. A *meta-edge* is drawn between two nodes v_1 and v_2 (where either or both are collapsed cluster nodes), if an edge exists between two nodes u_1 and u_2 such that $u_1 \in D(v_1)$ and $u_2 \in D(v_2)$ in the cluster hierarchy. This preserves the connectivity relationships between the clusters in a natural way. Figure 2.7 shows an example abridgement of the clustered graph shown in Figure 2.6. This definition of abridgements is effectively the same as in Huang's work [76], and is similar to the concepts of a *visual précis* and *horizon* presented by Quigley [127].

Clustered graphs can be used in many of the applications that incorporate graphs. This is often achieved by supplementing the graph with additional hierarchical grouping information, which results in a richer model and visualisation of the data. For example, the chemical reactions in metabolic pathways may be modelled at multiple levels of detail: the lowest level of detail would describe the full chemical reaction at an atomic scale, whereas the highest level would involve only compound reactions involving larger entities, such as proteins or enzymes. Fund manager flow graphs may recursively group nodes according to a hierarchy of market sectors, thus allowing analysis of the overall flow between markets, or the flow between each of the fund-managers in a market sector with the other market sectors. However, it is usually the case that this hierarchical grouping information is not able to be easily and clearly represented in standard (non-clustered) graph models. For instance, the hierarchical grouping could be represented by edges of a different type or colour, but such a representation is not as clear as a clustered graph.



(a) Original clustered graph

(b) Abridgement



(c) Cluster hierarchy

Figure 2.7: An example of an abridgement of the clustered graph from Figure 2.6.

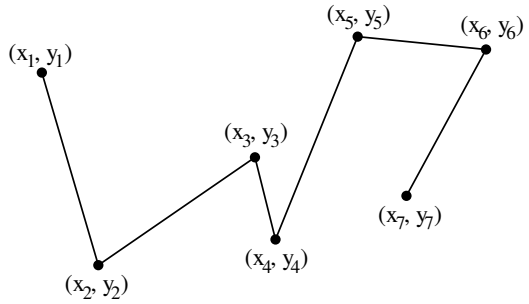


Figure 2.8: A polyline edge layout.

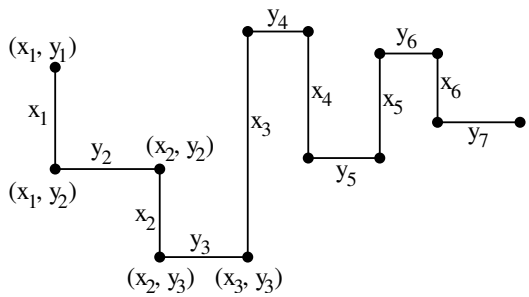


Figure 2.9: An orthogonal edge layout.

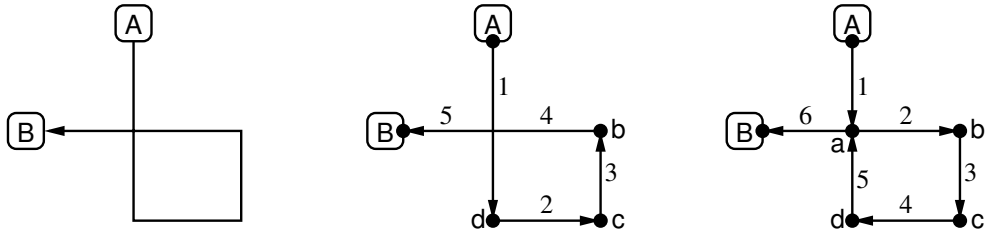
Edge layouts

Edges are usually visually represented by lines connecting the adjacent nodes. The lines may be composed of a single straight line segment, several line segments connected in a chain (a *polyline*), or may be more sophisticated, such as spline curves. In all of these cases, the edge layout is characterised by a series of *control points*. These control points are called *bends* for polyline edge layout, and the straight-line edge layout case can be considered to be a special case of the polyline edge layout where the series of bends is empty. Since the use of curved edges can considerably increase the sophistication required for edge handling, the scope of this study is restricted to polylines.

A polyline edge layout, such as shown in Figure 2.8, is represented by a sequence of points in the plane

$$((x_1, y_1), (x_2, y_2), ..., (x_n, y_n)).$$

The straight-line segments connecting bend points are referred to as *edge segments*. An *orthogonal edge layout* is a polyline edge layout in which all the edge segments are parallel to either the x axis (*horizontal segments*) or y axis (*vertical segments*), as shown in Figure 2.9. Orthogonal edge



(a) Normal (b) Self-intersecting (c) Self-occluding

Figure 2.10: An example of an ambiguous orthogonal edge layout between nodes A and B with self-intersecting and self-occluding possible actual layouts. In (b) the order of the bends is AdcbB, and in (c) the order of the bends is AabcdaB. The segment coordinates in (b) are $(x{=}0, y{=}0, -2, 1, -1, -1)$, whereas in (c) they are $(x{=}0, y{=}0, -1, 1, -2, 0, -1, -1)$.

layouts have the form

$$((x_1, y_1), (x_1, y_2), (x_2, y_2), (x_2, y_3), (x_3, y_3), ..., (x_n, y_n)),$$

$$\text{or} \quad ((x_1, y_1), (x_2, y_1), (x_2, y_2), (x_3, y_2), (x_3, y_3), ..., (x_n, y_n)),$$

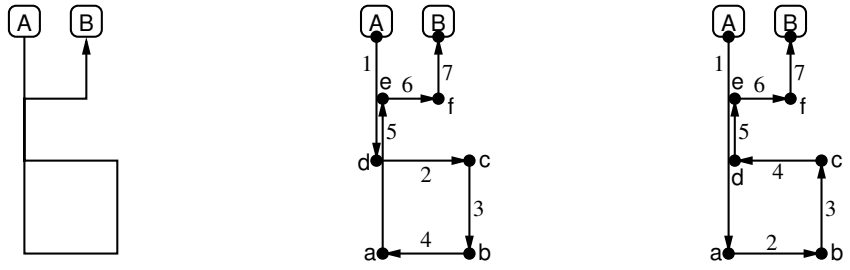
which is abbreviated to

$$(x{=}x_1, y{=}y_1, x_2, y_2, x_3, y_3, ..., x_n, y_n),$$

where it is understood that this ordered list of coordinates alternates between x and y coordinates. An orthogonal edge layout with k segments is of *length* k, and is specified by $k + 2$ coordinates. This means that the set of all orthogonal edge layouts of length k is $\mathbb{R}^{k+2}$.

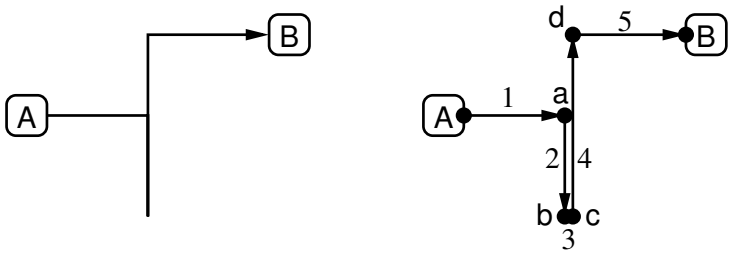
An orthogonal edge layout is said to be *ambiguous* if it is visually indistinguishable from another orthogonal edge layout (ignoring direction), or if it appears that its segments do not alternate in x and y. This ambiguity arises from the visual representation of an edge layout as a sequence of connected straight-line segments. For clarity, when discussing ambiguous orthogonal edge layouts, small circles are drawn at bends, arrowheads drawn on edge segments, and the segments are labelled with numbers indicating their order.

An orthogonal edge layout is *self-intersecting* if two or more edge segments intersect, and is *self-occluding* if two or more bends are coincident, or two or more segments share an intersection greater than a single point. An orthogonal edge layout is ambiguous if it is self-intersecting or self-occluding. Examples of ambiguous orthogonal edge layouts are shown in Figures 2.10, 2.11 and



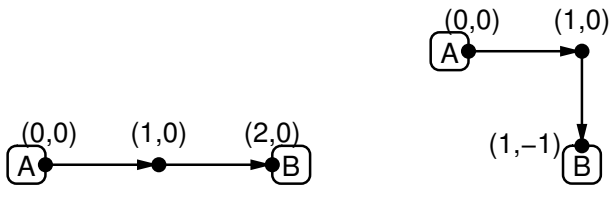
(a) Normal (b) Alternate 1 (c) Alternate 2

Figure 2.11: An example of an ambiguous orthogonal edge layout between nodes A and B with two self-occluding possible actual layouts. In (b) the order of the bends is AdcbaefB, and in (c) the order of the bends is AabcdefB. The segment coordinates in (b) are $(x{=}0, y{=}0, -2, 1, -3, 0, -1, 1, 0)$, whereas in (c) they are $(x{=}0, y{=}0, -3, 1, -2, 0, -1, 1, 0)$.



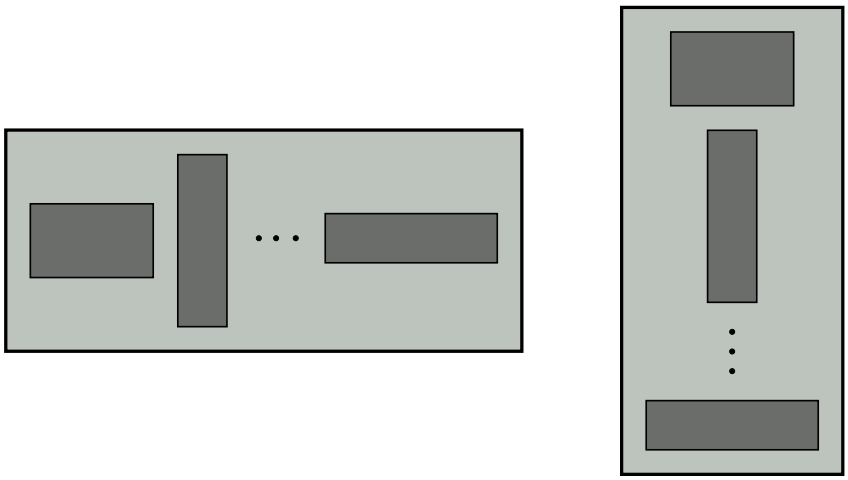
(a) Normal (b) Alternate

Figure 2.12: An example of an ambiguous orthogonal edge layout between nodes A and B. In (b) the order of the bends is AabcdB, where b and c are the same point, causing the 3rd segment to have 0 length (and thus be degenerate). The segment coordinates are $(x{=}0, y{=}0, 1, -1, 1, 1, 2)$.



(a) (b)

Figure 2.13: Two degenerate orthogonal edge layouts. The edge in (a) has value $(x{=}0, y{=}0, 1, 0, 2)$, that is, bends at points $((0,0),(1,0),(1,0),(2,0))$. Note the doubly repeated bend at $(1,0)$. The edge in (b) has value $(x{=}0, y{=}0, 1, 0, 1, -1)$, that is, bends at points $((0,0),(1,0),(1,0),(1,0),(1,-1))$. Note the triply repeated bend at $(1,0)$.



(a) Horizontal (b) Vertical

Figure 2.14: Horizontal and vertical inclusion tree layouts.

2.12.

An orthogonal edge layout is *degenerate* if it has any segments of zero length and is not ambiguous. Figure 2.13 shows two examples of this.

2.2 Layout

This section describes the approaches taken for the automatic layout and drawing of trees and clustered graphs. The general strategy is to solve the problem for trees in the inclusion layout style, and then to consider the layout or *routing* of clustered graph edges in the presence of the inclusion cluster hierarchy. This is because the focus of this thesis is on the strategy used for navigation, rather than the particular layout algorithms and techniques used.

2.2.1 H-V layout

An *h-v layout* is an inclusion tree layout in which all sibling nodes are aligned either in x or y, that is, either horizontally or vertically, as shown in Figure 2.14. This section reviews the h-v layout material presented in [44], including algorithms, strategies and results.

Inclusion layout style

The minimum inclusion layout problem described in Section 2.1.1 is NP-hard even for binary trees [44]. By restricting the possible arrangements of nodes, it is possible to use polynomial time binary tree algorithms that may be easily extended to general trees. Section 2.2.2 shows a polynomial time approach for MILP which removes the h-v constraint.

The simplest h-v layout algorithm is the greedy algorithm. This traverses the tree in a post-order fashion, from the leaves up to the root, and for each node chooses either horizontal or vertical arrangement based on which is locally better in terms of a given size measure. Since this decision is made once for each node in the traversal, the greedy algorithm has a linear running time. As is often the case with such algorithms, it is easy to find counter-examples illustrating how the greedy algorithm may make globally poor decisions based on local conditions. For example, Figure 2.15 shows a simple counter example of a tree where the greedy algorithm (using the area size measure) is globally sub-optimal.

An obvious optimal algorithm works by storing a list of the possible solutions for each non-leaf node. Each *possible solution* is a complete arrangement of the descendants of that node, specified recursively in terms of the possible solutions of the child nodes. That is, if the set of possible solutions for a node v is denoted by $L(v)$, we have

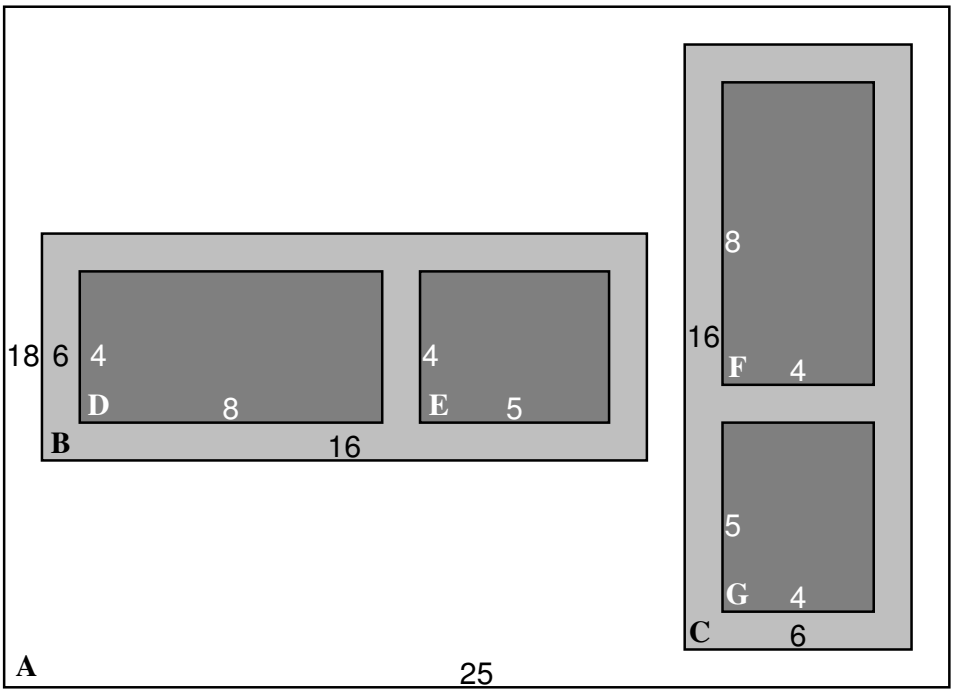
$$\begin{aligned} L(v) \quad = \quad & \{H(l_1, l_2, ..., l_m) \mid \forall\ l_i \in L(u_i),\ \forall\ u_i \in C(v)\,\} \cup \\ & \{V(l_1, l_2, ..., l_m) \mid \forall\ l_i \in L(u_i),\ \forall\ u_i \in C(v)\,\}, \end{aligned}$$

where the function H indicates the possible solution obtained by horizontal arrangement of the specified possible layouts, and similarly for the function V and a vertical arrangement.

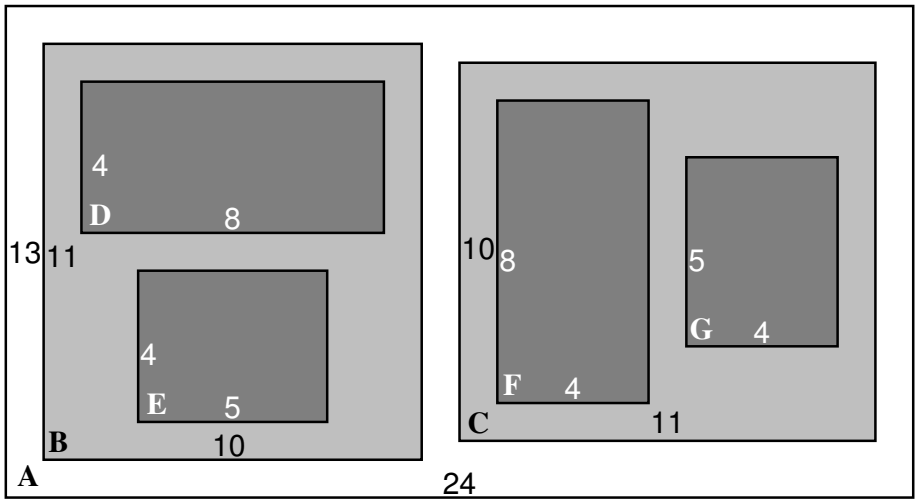
The algorithm proceeds by building these lists in a post-order fashion from the leaves up to the root node. When completed, the root node has a list, $L(r)$, of all possible overall solutions that are available. This list can be traversed to find the best solution according to a given size measure,

$$l_{\min} = l \in L(r) \text{ such that } \psi(l) < \psi(l')\ \ \forall\ l' \in L(r), l' \neq l$$

This overall solution specifies a horizontal or vertical arrangement for every non-leaf node, which the algorithm can implement from the leaves upward (since the leaf sizes are fixed), in order to obtain a drawing of the solution.



(a) Greedy solution



(b) Better solution

Figure 2.15: An example of the sub-optimality of the greedy approach to the minimum inclusion layout problem. Node dimensions are indicated, gaps between nodes and margins inside non-leaf nodes have size 1. In (a), the overall area of node A is $25 \times 18 = 450$, whereas in (b) it is $24 \times 13 = 312$. Nodes B and C are locally optimal in (a), with area of $6 \times 16 = 96$, compared to an area of $10 \times 11 = 110$ in (b). The symmetry of the leaf nodes means that the area of A is the same irrespective of the decision to arrange nodes B and C horizontally or vertically (in both (a) and (b)).

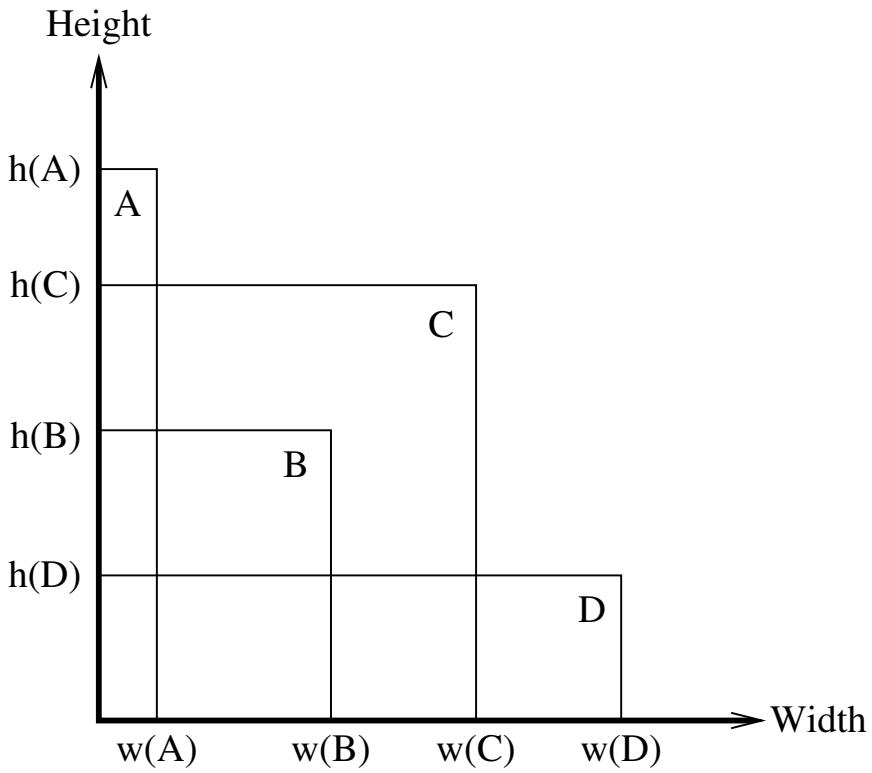
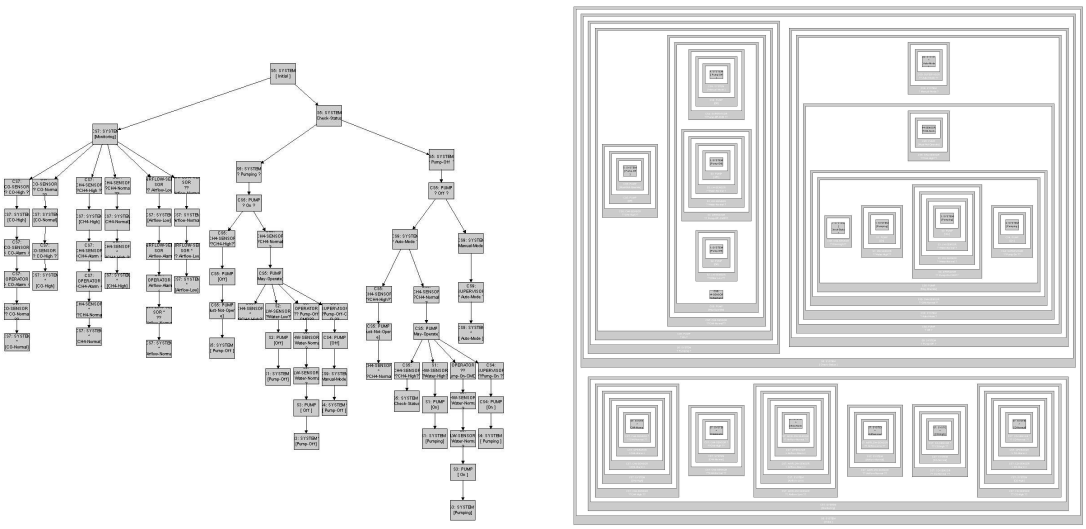


Figure 2.16: Dominating rectangles, where the width of the rectangle is associated with the x value, and the height with the y value. Rectangle C dominates B. Rectangles A, B, D are non-dominating, rectangle C is dominating.

However, such an exhaustive search is clearly inefficient in both space and time, since the number of possible solutions for a node grows exponentially with the number of descendant nodes. The concepts of *dominating rectangles* and *non-decreasing size measures* are used to reduce the number of solutions to a polynomial. A rectangle (c, d) is said to *dominate* another rectangle (a, b) if and only if $c \geq a$ and $d \geq b$, as shown in Figure 2.16. A size measure is *non-decreasing* in both dimensions if $\psi(a, b) > \psi(c, d)$ whenever $a > c$ and $b > d$. All the size measures presented in Section 2.1.1 are non-decreasing. If a possible solution l_1 of size (c, d) dominates l_2 of size (a, b), l_1 can never be included in the optimal layout for a non-decreasing size measure, since it could be replaced by l_2 to give a smaller layout. Thus, dominating possible solutions such as l_1 are discarded. It is easy to modify the optimal algorithm to store only non-dominating possible solutions, rather than all possible solutions. Further, the possible solutions are stored in order of increasing x: note that the definition of dominating rectangles means that the x coordinates are unique and the list is also decreasing in y. For example, in Figure 2.16 the rectangles A, B, D could be a list of non-dominating possible solution sizes.



(a) Node-link (b) Inclusion

Figure 2.17: The Mine Pump DBT using a node-link and inclusion layout style. This is obtained by using the optimal polynomial time algorithm presented.

When constructing the list of possible solutions for a node, the lists of the children can be "merged" in a fashion similar to the merge sort algorithm, since they are already sorted. Extending the treatment for binary trees presented by Eades *et. al.* [44], the algorithm maintains a list of pointers into the lists of possible solutions for each child. The composition of the current possible solutions is allowed only if it does not dominate the previously added solution. The appropriate pointers are advanced in the same way as the merging of many sorted lists, and the next composition is considered. This entire process is performed separately for horizontal and vertical arrangements, and the results then merged again to obtain the final list of possible solutions for the node.

The number of possible solutions for any node, using this technique, is shown to be polynomial by [44], specifically

$$O\left(\sum_{v \in L} d_v X_v\right),$$

where L is the set of leaf nodes, d_v is the depth of node v and X_v is the integer width of node v. This result requires integer leaf node sizes, and also depends on the widths of the leaf nodes. Neither of these restrictions are problematic in practice.

Figure 2.17(b) shows the result of applying this inclusion tree layout algorithm to the Mine Pump Design Behaviour Tree (DBT) in Figure 2.17(a). DBTs are discussed in Chapter 5.

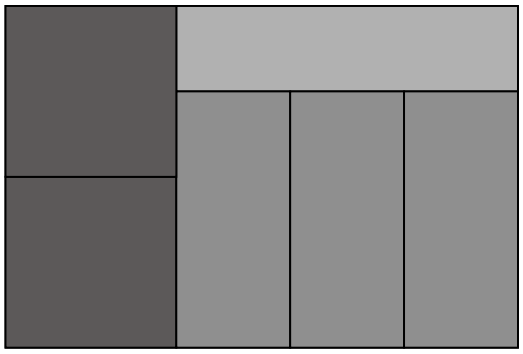
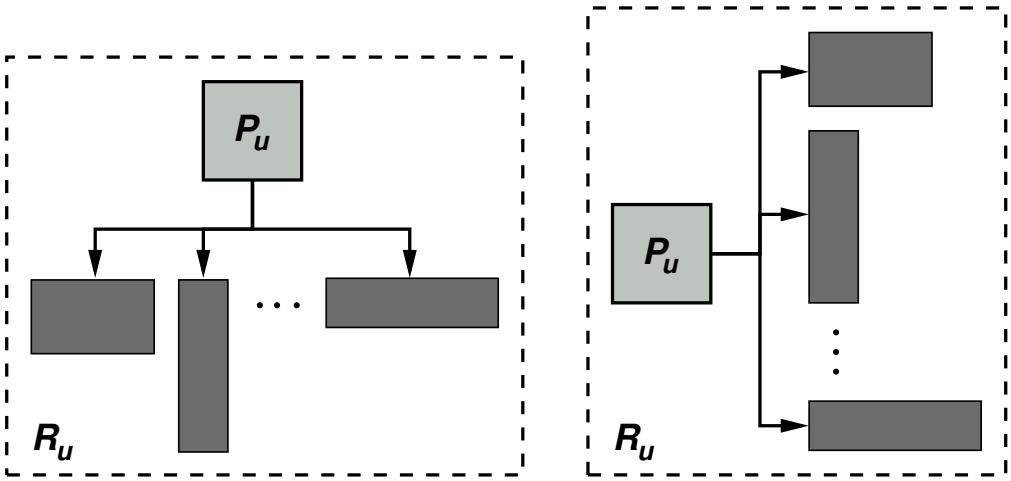


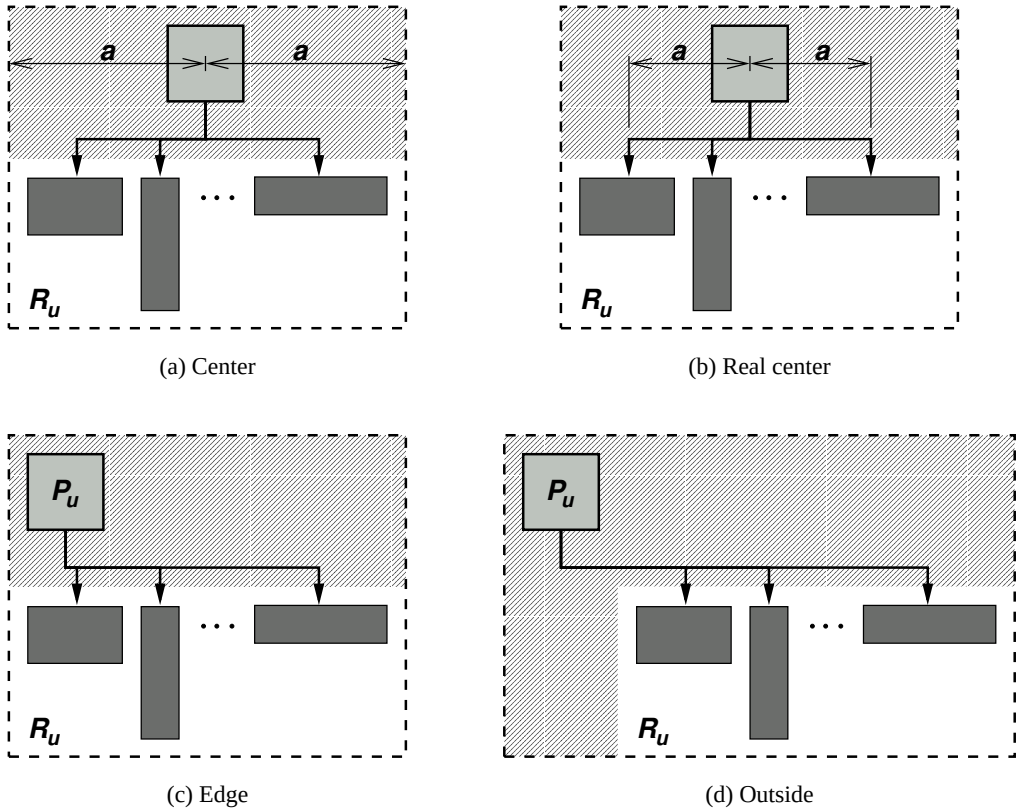
Figure 2.18: Treemap for the tree shown in Figure 2.1.



(a) Horizontal (b) Vertical

Figure 2.19: Horizontal and vertical tip-over tree layouts.

The h-v inclusion layout style is similar to *treemaps* [81], a space-filling technique for drawing trees in the plane. Figure 2.18 shows an example treemap of the tree shown in Figure 2.1. Treemaps tend to be used more commonly where some statistical or scalar data is associated with each node, and treemap algorithms are geared towards using and showing this data when computing the layout. This thesis uses inclusion layouts because the emphasis is on the structure of the nodes. However, since treemaps can be considered to be inclusion trees with no margins around the non-leaf nodes, the ideas presented in this thesis may also be applied to treemaps.



(a) Center (b) Real center

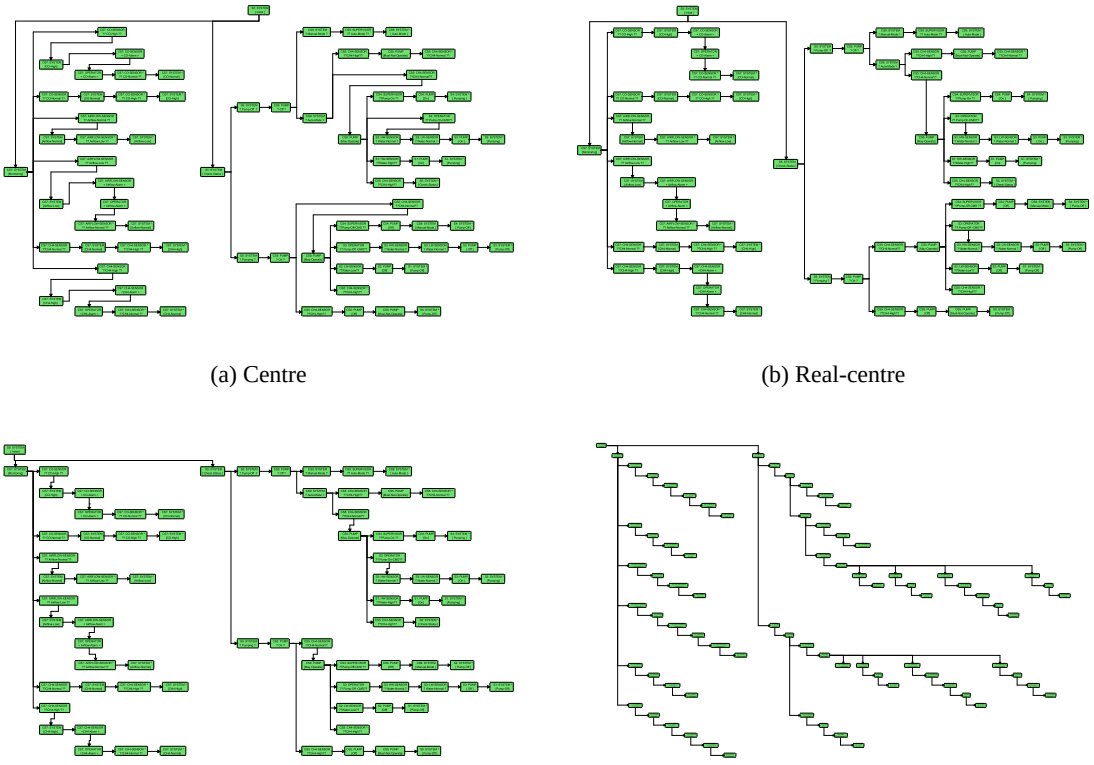
(c) Edge (d) Outside

Figure 2.20: Possible parent placement strategies. The hatched regions indicate the additional space caused by the tip-over style, and the regions with white background indicate the area required by the h-v inclusion layout. In (a) and (b), a is used to indicate equal lengths.

Node-link layout style

It is possible to adapt the h-v inclusion layout algorithm to the node-link style, giving an *h-v node-link* or *tip-over* layout style. The algorithm is identical, and the solution requires a choice of horizontal or vertical arrangement for each node. However, the size of a node is computed in a slightly different fashion, because additional space must be included for the node-link display of the parent node. Figure 2.19 shows the horizontal and vertical tip-over layout arrangements, corresponding to the h-v inclusion arrangements in Figure 2.14, with the R_u region used by the algorithm when considering the size of node u. The rectangle P_u is the visual representation of the node u, as opposed to the enclosing container used in the inclusion layout style, and the edges are drawn as indicated.

There are several possible ways to place the parent node P_u, most notably the "centre", "real-centre", "edge" and "outside" strategies. These parent placement strategies are illustrated for horizontal arrangement in Figure 2.20 with the additional regions they induce (compared to the region



(a) Centre (b) Real-centre

(c) Edge (d) Outside

Figure 2.21: The Mine Pump DBT from Figure 2.17 in tip-over layout style, for each of the four parent placement strategies shown in Figure 2.20.

required by inclusion layout). The corresponding vertical arrangements are analogous. The *centre* parent placement strategy places the parent halfway across the width of R_u. The *real-centre* parent placement strategy places the parent halfway between the midpoints of the far-left and far-right child nodes. The *edge* parent placement strategy aligns the left edges of the parent node and first child node. The *outside* parent placement strategy places the parent to the left of first child node. The tip-over layout presented in [44] uses only the 'centre' strategy for horizontal nodes, and the 'outside' strategy for vertical nodes.

Figure 2.21 shows the tree from Figure 2.17 in tip-over layout style, using each of the four parent placement strategies. Figure 2.21(d) clearly shows that the 'outside' strategy is a very poor parent placement strategy, compared to the other three. It has extra additional space in both dimensions, and guarantees at least one edge-bend for every node. This gives a layout which is much larger and harder to read, particularly for paths of nodes with degree of 1. Of the remaining three strategies,

'real-centre' appears to be the best. This is because it avoids unnecessary edge bends, especially with paths of degree 1, and nodes with alternating horizontal/vertical arrangements, which are particularly bad in the 'centre' strategy. The 'centre' strategy also suffers from ambiguity arising from the parent node possibly being considerably far away from the child nodes, such as with the root node in Figure 2.21(a).

2.2.2 Arbitrary node layout

The primary problem with the h-v layouts presented in the previous section is that they do not scale well as the degree of nodes increases. This is because, as the degree increases, both of the horizontal and vertical choices necessarily result in layouts with extreme aspect ratios. We demonstrate this with the following lemma and theorem:

Lemma 2.2.1 *There exists a tree T with n nodes and maximum degree d such that*

$$\psi(R_{\mathrm{hv}}(r)) = O(d) = O(n)$$

where ψ is the smallest enclosing square size measure and $R_{\mathrm{hv}}(r)$ is the region occupied by the root node in the optimal h-v layout.

Consider the tree (shown in Figure 2.22) of two groups of m nodes, each of unit width and height. Each group of nodes is connected to a common parent node, and these common parents are in turn connected to the root node. This gives $n = 2(m+1)+1$ nodes and maximum degree $d = m$, thus $d = O(n)$. Figure 2.23 shows the three possible h-v inclusion layouts of this tree (up to isomorphism). These layouts have sizes of $(2, m)$, $(m+1, m)$ and $(2m, 1)$, respectively (ignoring the constant margin factors between nodes). This gives size measures ψ of m, $m + 1$ and $2m$ respectively. Thus the optimal h-v layout is the first, as shown in Figure 2.23(a), with size measure $\psi(R_{\mathrm{hv}}(r)) = m = d = O(d) = O(n)$.

Theorem 2.2.2 *There exists a tree T with n nodes such that*

$$\frac{\psi(R_{\mathrm{hv}}(r))}{\psi(R_{\mathrm{opt}}(r))} \geq O(\sqrt{n})$$

where ψ is the smallest enclosing square size measure, $R_{\mathrm{hv}}(r)$ is the region occupied by the root node in the optimal h-v layout and $R_{\mathrm{opt}}(r)$ is the region occupied by the root node in the optimal inclusion tree layout. That is, a tree exists for which the h-v is sub-optimal by a factor of $O(\sqrt{n})$.

Again, consider the tree shown in Figure 2.22. The layout shown in Figure 2.24(a) places each group of m nodes into a matrix of $\sqrt{m/2} \times \sqrt{2m}$. Thus, the overall layout has size $(\sqrt{2m}, \sqrt{2m})$, giving a size measure of $\sqrt{2m}$, which is optimal. From Lemma 2.2.1 we know that $\psi(R_{\mathrm{hv}}(r)) = m$, and

$$\frac{\psi(R_{\mathrm{hv}}(r))}{\psi(R_{\mathrm{opt}}(r))} = \frac{m}{\sqrt{2m}} = \sqrt{\frac{m}{2}} = O(\sqrt{n})$$

This completes the proof.

In fact, this can still be true for some non-optimal layouts, such as that shown in Figure 2.24(b). This arranges each group of m nodes into a matrix of size $\sqrt{m} \times \sqrt{m}$, giving an overall layout of size $(2\sqrt{m}, \sqrt{m})$ and size measure of $2\sqrt{m}$. Thus, the result still holds:

$$\frac{\psi(R_{\mathrm{hv}}(r))}{\psi(R_{\mathrm{opt}}(r))} = \frac{m}{2\sqrt{m}} = \frac{\sqrt{m}}{2} = O(\sqrt{n})$$

Layouts in which the h-v property is not preserved, such as in Figures 2.24(a) and 2.24(b), are called *arbitrary inclusion layouts*.

The problem of large degree nodes can arise in real-world situations. Figure 2.25 shows the inheritance tree structure for the core Java 2 language (version 1.4.1) [66] with 773 nodes, in a node-link style produced using the Sugiyama-based tree drawing algorithm [152] from the Tom Sawyer Visualization 6.0 Java Edition software [158]. It shows one node of very high degree, `java.lang.Object`, which occurs as a consequence of the design of the Java language. This node has 267 children, where the rest of the non-leaf nodes have an average of 3.6 children and a standard deviation of 8.4 children. This indicates that there are very few nodes of high degree, and most nodes (over 75%) have fewer than the average number of children.

Figure 2.26 shows the optimal h-v layout of this tree (using the minimum enclosing square size measure). Figure 2.27 shows an alternative layout for this tree, this time an arbitrary inclusion layout. The arbitrary layout is considerably smaller and has less “wasted space” than the h-v layout. This is despite the fact that the h-v layout in Figure 2.26 is an optimal h-v layout, whereas the

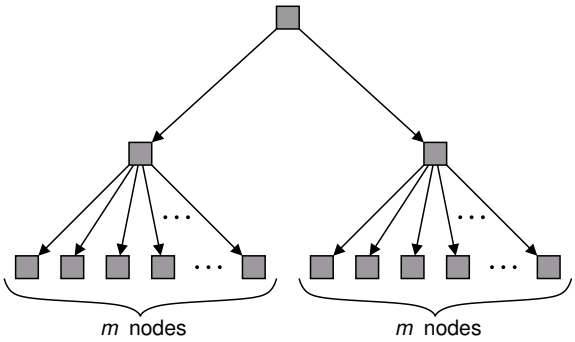
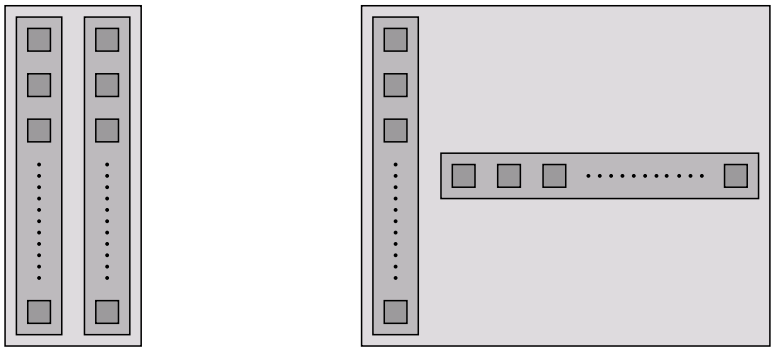
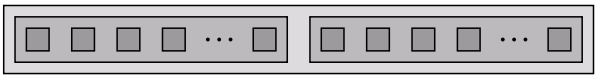


Figure 2.22: Two groups of m nodes, each connected to a common parent node.

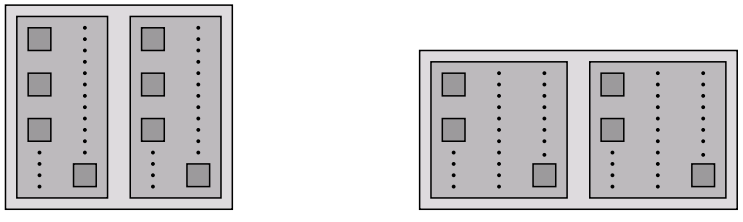


(a) $R(r) = (2, m)$ (b) $R(r) = (m + 1, m)$



(c) $R(r) = (2m, 1)$

Figure 2.23: The three possible h-v inclusion layouts of the tree from Figure 2.22 (up to isomorphism).



(a) Optimal, $R(r) = (\sqrt{2m}, \sqrt{2m})$ (b) Sub-optimal, $R(r) = (2\sqrt{m}, \sqrt{m})$

Figure 2.24: The optimal and a sub-optimal arbitrary inclusion layout of the tree from Figure 2.22. Both of these layouts improve on the optimal h-v layout by a factor of $O(\sqrt{n})$.

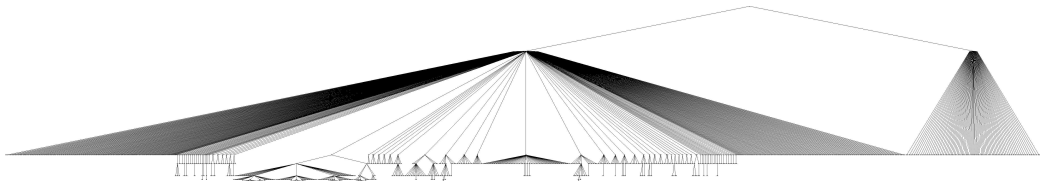


Figure 2.25: The inheritance tree structure for the core Java 2 language, in a node-link tree style.

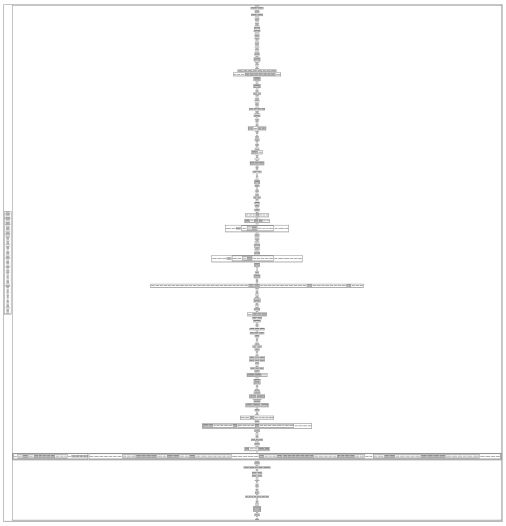


Figure 2.26: The inheritance tree structure for the core Java 2 language, using the h-v MILP inclusion layout algorithm. It has dimensions 22185×22992 (in arbitrary units).

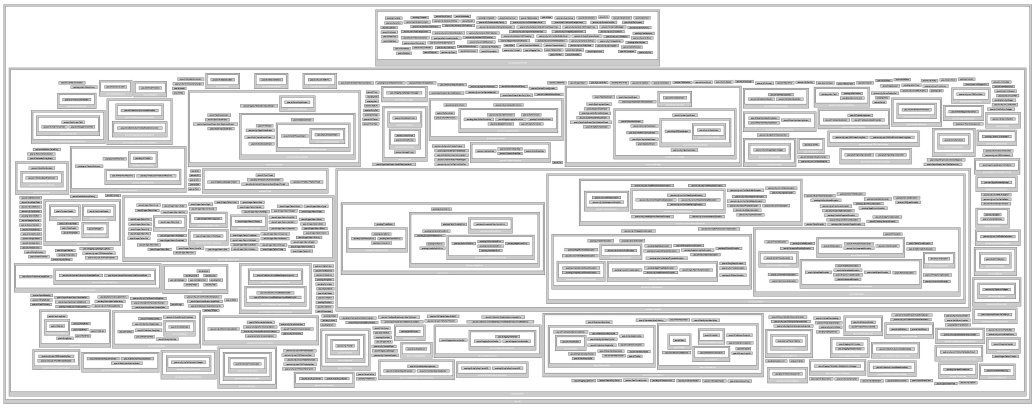


Figure 2.27: The inheritance tree structure for the core Java 2 language, using a better inclusion layout algorithm. It has dimensions 8818×2688 (in the same units as Figure 2.26).

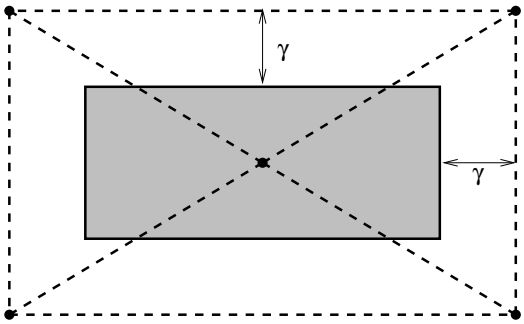


Figure 2.28: The initial state of the JBILA layout region and layout triangulation after adding the first node.

arbitrary layout in Figure 2.27 is not necessarily optimal. This shows a real-world example of where nodes of large degree can arise, and the inadequacy of h-v layouts in such cases.

Arbitrary node layout algorithms

Now that the inadequacy of the h-v layout has been discussed, attention is turned to efficient arbitrary inclusion layout algorithms.

The original binary tree h-v inclusion layout [44] incorporates a treatment for drawing labels on each node. These labels are considered to be additional nodes, thus effectively creating a ternary tree. The layout of this is then solved as for binary h-v trees, with the addition of some extra cases for placing the "label" node adjacent to the actual nodes, rather than in horizontal or vertical alignment with them. This solution, though, is still basically an h-v solution. The h-v constraint has been relaxed slightly, and it could be relaxed further, for example, by allowing two or more "rows" or "columns" of h-v laid out nodes. However, this solution is still not as general as is desired. In fact, it even introduces new problems, such as how to best partition the nodes into rows/columns and how many rows/columns should be used. These problems are non-trivial, so the decision to avoid this approach is a matter of practicality as well as elegance.

A better, more general solution to the MILP, using arbitrary layouts, is presented by Itoh *et al.* in their *Jewellery Box Inclusion Layout Algorithm* (*JBILA*) [80]. This algorithm was used to create the layout shown earlier in Figure 2.27.

The algorithm works by processing each non-leaf node in a depth-first, post-order traversal, as with previous algorithms. For each non-leaf node the child nodes are placed one at a time, in decreasing order of size. As in Section 2.1.1, the size of a node is defined according to the smallest enclosing rectangle size measure. The first node is simply placed at the origin $(0, 0)$, and the *layout*

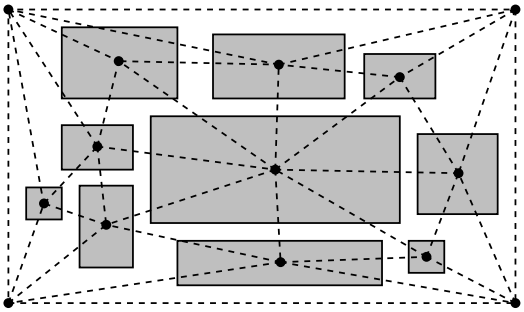


Figure 2.29: An example of the JBILA layout region and layout triangulation during the algorithm.

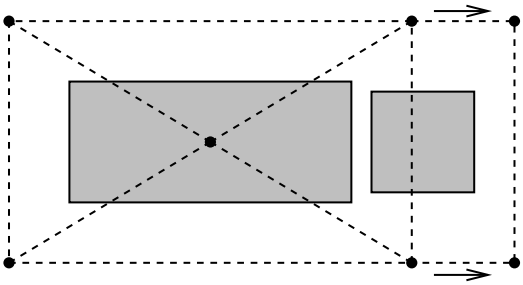


Figure 2.30: An example of extending the layout region.

region is defined as the region enclosing the first node with an additional margin of size γ, as shown in Figure 2.28. The *layout triangulation* consists of a *Delaunay triangulation* [49, 112] with vertices at the corners of the layout region and at the centre of each node. The algorithm attempts to place each node at a position within the layout region such that it does not overlap any previously placed nodes and it is wholly within the layout region. If this is not possible, then the algorithm places the node at a position where it does not overlap with any previously placed nodes, and extends the layout region to accommodate the placement of the node. The layout triangulation is adjusted, as necessary, by moving the layout region corner vertices, adding a vertex at the center of the added node, and then updating the triangulation to ensure that it is still a Delaunay triangulation. Figure 2.29 shows an example of the JBILA layout region and layout triangulation after placing several nodes.

When attempting to place each node, the algorithm tries a sequence of *candidate points* for placement of the node. Candidate points which do not cause the node being placed to overlap with any previously placed nodes, are said to be *valid candidate points*. Candidate points which cause the node being placed to be contained within the layout region, are said to be *interior candidate points*. The algorithm attempts to place the node at the first candidate point which is both valid and

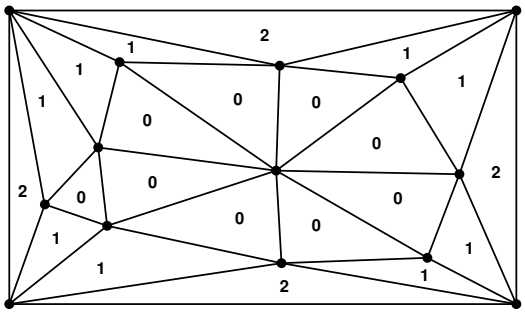


Figure 2.31: An example of the interior number in the layout triangulation.

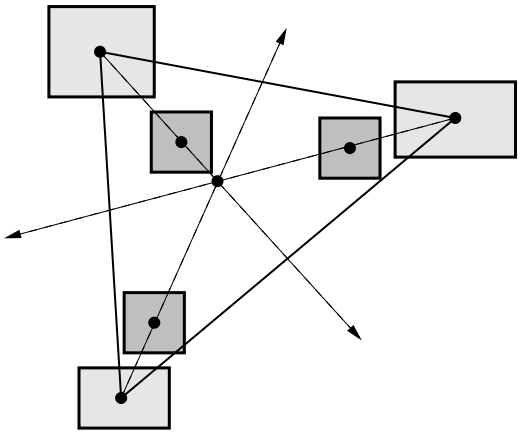


Figure 2.32: Candidate points are placed along rays from each triangle corner node through the triangle centre.

interior. If no such candidate point exists, the algorithm places the node at the first valid candidate point, extending the layout region as illustrated in Figure 2.30. As shown below, a valid candidate point always exists.

Candidate points are found by searching through triangles in the layout triangulation. Triangles are searched in order of their "interior number" and size. The *interior number* c is defined as the number of vertices of the triangle which are corners of the layout region. This gives possible values $c = 0, 1, 2, 3$, as shown in Figure 2.31[3]. The algorithm first searches within the group of $c = 0$ triangles, then the $c = 1$, $c = 2$ and $c = 3$ triangles. This heuristic attempts to avoid extending the layout region. The group of triangles with the same c value are considered in decreasing order of size. This is because it is easier to place nodes in larger triangles than smaller ones. Here the size of a triangle is considered to be its area.

Within each triangle, the algorithm constructs *candidate rays* from each corner point through the

[3] Although $c = 3$ never occurs during the algorithm.

centre of the triangle. The centre of a triangle is taken as the centroid, although using the incentre is also possible. Neither the circumcentre nor the orthocentre of the triangle are useful, as they do not lie in the interior of obtuse triangles. A candidate point is found on each candidate ray by placing the node adjacent to the corner node, and with its centre on the candidate ray. This is shown in Figure 2.32. Candidate points that lie on the line between the corner and centre of the triangle are considered before those that do not. The use of rays guarantees that at least one candidate point can be found, even if it is located away from the rest of the nodes (causing the layout region to expand considerably).

Of course, it is not desirable to have adjacent nodes touching one another; an *inter-node gap* of δ is required. This is easily achieved by treating the size of node being placed to be expanded by δ on each side.

The time complexity of the overall algorithm is dominated by the time taken to construct the Delaunay triangulation, which is $O(n \log n)$, for n nodes. This time complexity is sufficiently small to allow the algorithm to be used interactively on large graphs (that is, thousands or perhaps tens of thousands of nodes).

Section 4.2.1 presents an extension to the Jewellery Box Inclusion Layout Algorithm that attempts to minimise changes with respect to a previous layout of the tree (or a similar tree).

2.2.3 Orthogonal edge routing

Sections 2.1.2 and 2.2 demonstrated that a clustered graph is an inclusion tree with the addition of edges between nodes. As such, an *edge routing algorithm* is required to determine the layout of the edges in the presence of the already placed cluster hierarchy inclusion tree. In this thesis, we are interested in the Structural Zooming technique itself, rather than the intricacies of the particular type of edge layout or edge routing algorithm used. Thus, in order to simplify the edge routing and the subsequent application of Structural Zooming to clustered graphs, described in detail in Chapter 4, only *orthogonal* edge layouts are considered here.

The *orthogonal edge routing algorithm* used is that provided by the Tom Sawyer Visualisation 6.0 Java Edition software [158]. It is a fast incremental layout algorithm which orthogonally routes the edges, and ensures that there are no edge ambiguities in the resulting layout. It allows control over whether node positions are fixed or free to move, in order to minimise crossings and bends, and whether node sizes are fixed or free to increase, in order to maintain minimum edge spacing requirements.

2.3 H-V layout size measure evaluation

This section presents the first contribution of the thesis, an empirical evaluation of the choice of size measure for h-v inclusion layouts. The four main size measures from Section 2.1.1 are compared, namely, area, perimeter, minimum enclosing square and square aspect ratio. These results were originally published by the author in Pulo and Takatsuka [124].

The properties used to empirically investigate the different inclusion layout size measures and algorithms are considered first. The most fundamental is the *non-dominating function plot*, which plots the list of all non-dominating possible solutions with the x and y coordinates as the width and height (respectively) of each possible solution.

Figure 2.33 illustrates this concept, while Figure 2.34 shows the the non-dominating function for the tree shown in Figure 2.1. Figure 2.34 also shows the sizes of all the possible solutions as small circles. Note, that the non-dominating function consists of the "lower-left frontier" of these points, as the other points dominate at least one of the points which lie on the non-dominating function. Strictly speaking, the points of the non-dominating function should not be joined in Figure 2.33, as it is not a continuous function. However, when the non-dominating functions of several trees are plotted on the same set of axes for comparison, drawing them as lines is more useful than disconnected dots.

The x and y coordinates for each inclusion tree layout have been normalised based on the largest x or y coordinate value. Consider the layout specified by the leftmost point on the non-dominated function: this is the *tallest non-dominated layout* (commonly the layout with all vertical arrangements). Similarly, the rightmost point is the *widest non-dominated layout*.

The *normalisation factor* is the larger of either: the height of the tallest non-dominated layout, or the width of the widest non-dominated layout. The corresponding layout is called the *normalisation layout*.

Figure 2.35 shows the different types of non-dominating functions possible. A general indication of the possible quality of the inclusion layout is given by the position and concavity of the function. Layouts near the origin $(0, 0)$ are preferable to those near $(1, 1)$ as they are relatively smaller (in comparison to the width or height of the normalisation layout). Layouts near the line $y = x$ are preferable to those near the points $(1, 0)$ and $(0, 1)$ as they have less extreme aspect ratios.

Trees which have non-dominating functions, similar to the "bad case" in Figure 2.35, are poor candidates for inclusion layout. This is because there is only a small variation between the possible

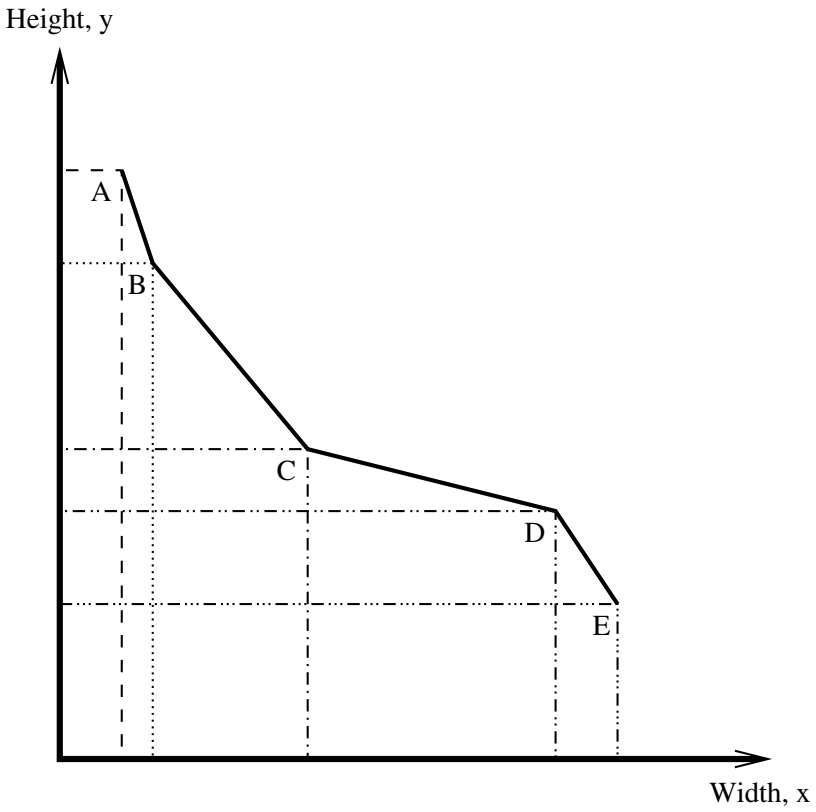


Figure 2.33: Non-dominating rectangles as a function. Note that as x values increase, the y values decrease (strictly) monotonically.

layouts and all have x and y dimensions similar to the maximum x or y dimension. The concavity indicates that the layouts, with aspect ratios closer to 1, have (relatively) larger x and y dimensions. By contrast, non-dominating functions similar to the "good case" are better candidates for inclusion layout, as their possible layouts have a larger variation of dimensions, with some layouts having x and y dimensions much smaller than the maximum x or y dimension (approaching the region near (0,0)).

The next property to be examined is how the different 'size measures' vary with the different non-dominating solutions. The third dimension is used to indicate the value of the chosen measure for each layout in the non-dominating function, as shown in Figure 2.36. The "best" layout, according to this measure, is the one with the smallest z value. However, this plot becomes harder to read when comparing several measures, or comparing the measures of several non-dominating functions, and so a two-dimensional variant is used as shown in Figure 2.37. The x axis of this plot is $x - y$ from Figure 2.36, and is thus parallel to the line $x + y = 1$ in Figure 2.34. The vertical dotted line at $x = 0$ corresponds to the line $y = x$ in Figure 2.34. The dynamic programming

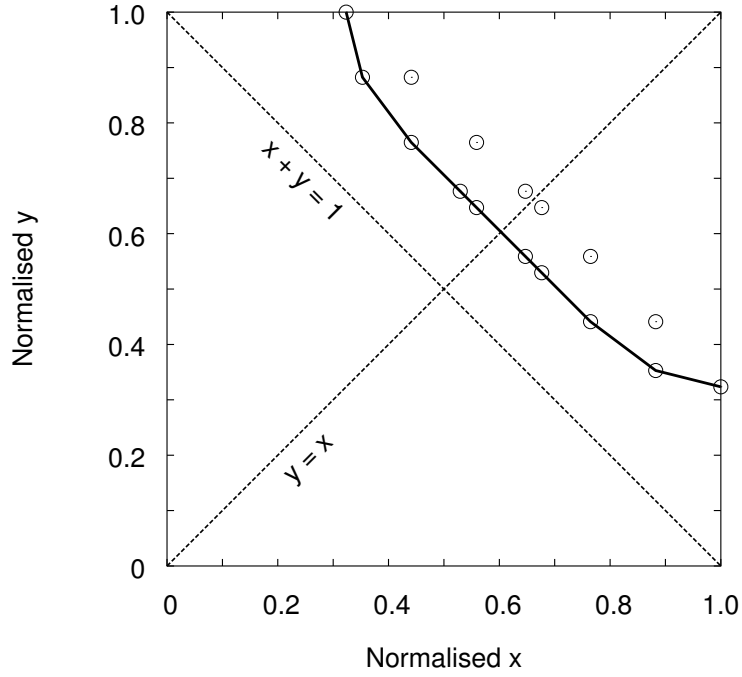


Figure 2.34: The non-dominating function plot for the tree in Figure 2.1. The non-dominating function is indicated by the solid line, while possible layout solutions are indicated by circles.

algorithm chooses the layout with the minimum value for a given measure. Animated Figure 2.38 shows how Figure 2.37 is obtained from Figure 2.36.

2.3.1 Results

This section presents the results of using the H-V inclusion layout style on two sources of real world trees.

- *Design behaviour trees* (*DBT*s) are a method for representing requirements in software engineering [36, 37], discussed in more detail in Chapter 5. The size of a DBT depends on the size of the software system it describes, and may range from 20 nodes to 500. Eleven DBTs are used in this investigation.

  A typical DBT is shown in node-link and inclusion styles in Figure 2.17. One observation about these trees is that they contain many nodes of degree 1, along with the occasional node of degree 4 or more.

- *Ontologies* are formal descriptions of concepts and relationships which exist in a

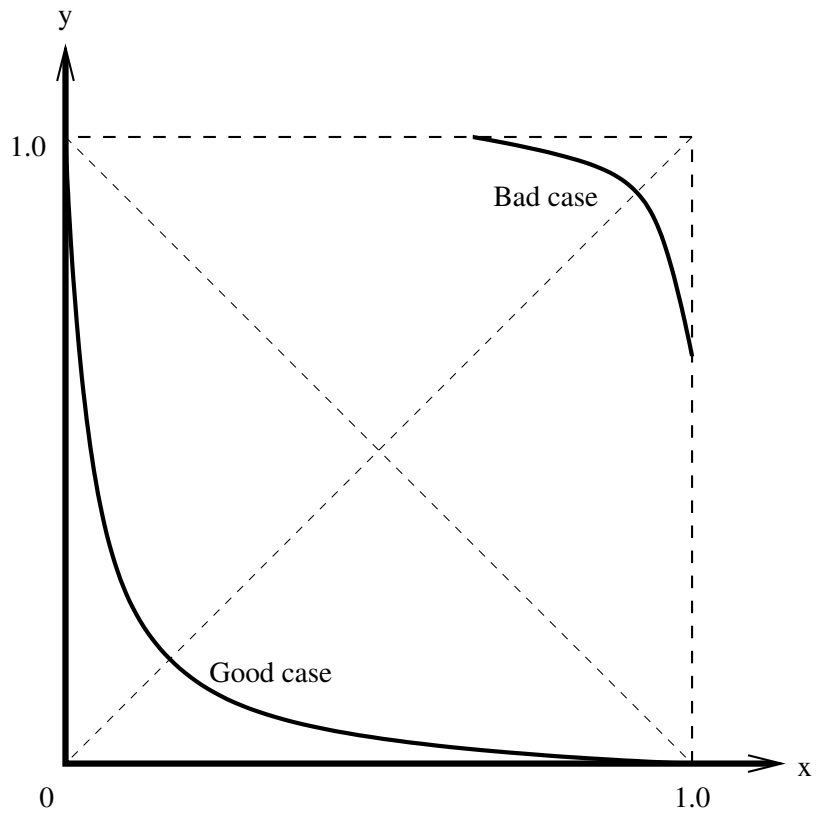


Figure 2.35: General quality of non-dominating functions.

given domain [69]. They are commonly used in artificial intelligence (AI) to support knowledge-sharing between various AI programs or agents. Among other things, they contain a class hierarchy for the various types of individuals defined in the ontology. This class hierarchy is sometimes called a *taxonomy*, and it is this tree which we consider for inclusion layout. Compared to DBTs, these trees tend to have fewer nodes of degree 1 and are generally broader. Two taxonomies are considered:

- The Multi-Sensory Taxonomy (MST) presented in Nesbitt's PhD thesis [107], with 290 nodes.
- The Enterprise ontology [164] from the Ontolingua server at the Stanford University Knowledge Systems Laboratory [46], with 95 nodes. Figure 2.39 shows the class hierarchy of this ontology in node-link and inclusion styles.

Figure 2.40(a) shows the non-dominating function plot (as shown in Figure 2.34), for all the input trees. Figures 2.40(b)–2.40(e) show the results of the four size measures for all the input trees

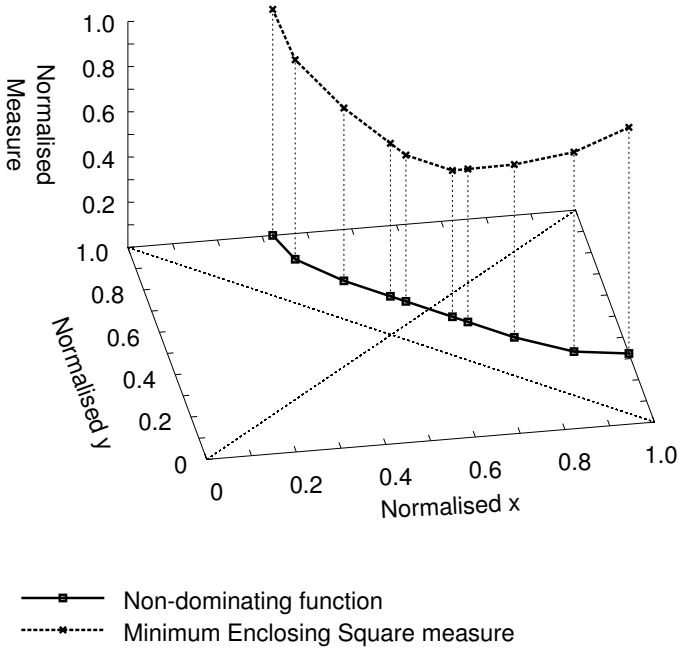


Figure 2.36: Plotting a size measure above the non-dominating function of Figure 2.34.

Tree	Number of	
	Nodes	Layouts
Integrated Low Level DBT	504	364
Multi-Sensory Taxonomy	290	216
Satellite Low Level DBT	269	468
Enterprise Ontology	95	41
Integrated High Level DBT	89	38
Mine Pump DBT	78	20
Online Shopping Low Lvl DBT	43	11
Online Shopping Med Lvl DBT	40	18
12207 Acquisition DBT	40	6
Satellite High Level DBT	33	15
Car System DBT	22	6
Online Shopping High Lvl DBT	21	2
Man Fishing DBT	17	2

Table 2.1: Sizes of the input trees (*Number Nodes*) and the number of non-dominating layouts (*Number Layouts*), shown in decreasing order of number of nodes.

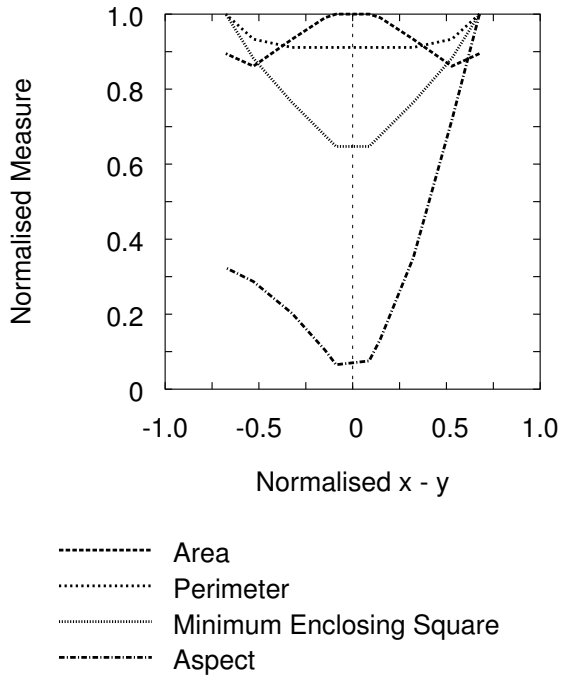


Figure 2.37: Various size measure plots for the tree in Figure 2.1.

(as shown in Figure 2.37). Table 2.1 lists the input trees together with the number of nodes and the number of non-dominating layouts.

In Figure 2.40(a) it can be seen that the position and concavity of the non-dominating functions varies considerably between the input trees. Those with the best position and concavity are the Multi-Sensory Taxonomy, Integrated High Level DBT, Enterprise Ontology, Integrated Low Level DBT and the Mine Pump DBT. From Table 2.1 we can see that these are also the largest trees in the input — with the notable exception of Satellite Low Level DBT. However, for tree-sizes less than about 60 nodes, the non-dominating functions tend to be in the upper-right half of the plot (that is, above the $x + y = 1$ line) and are relatively flatter. The figure of 60 nodes is only approximate, due to the uneven distribution of input tree sizes.

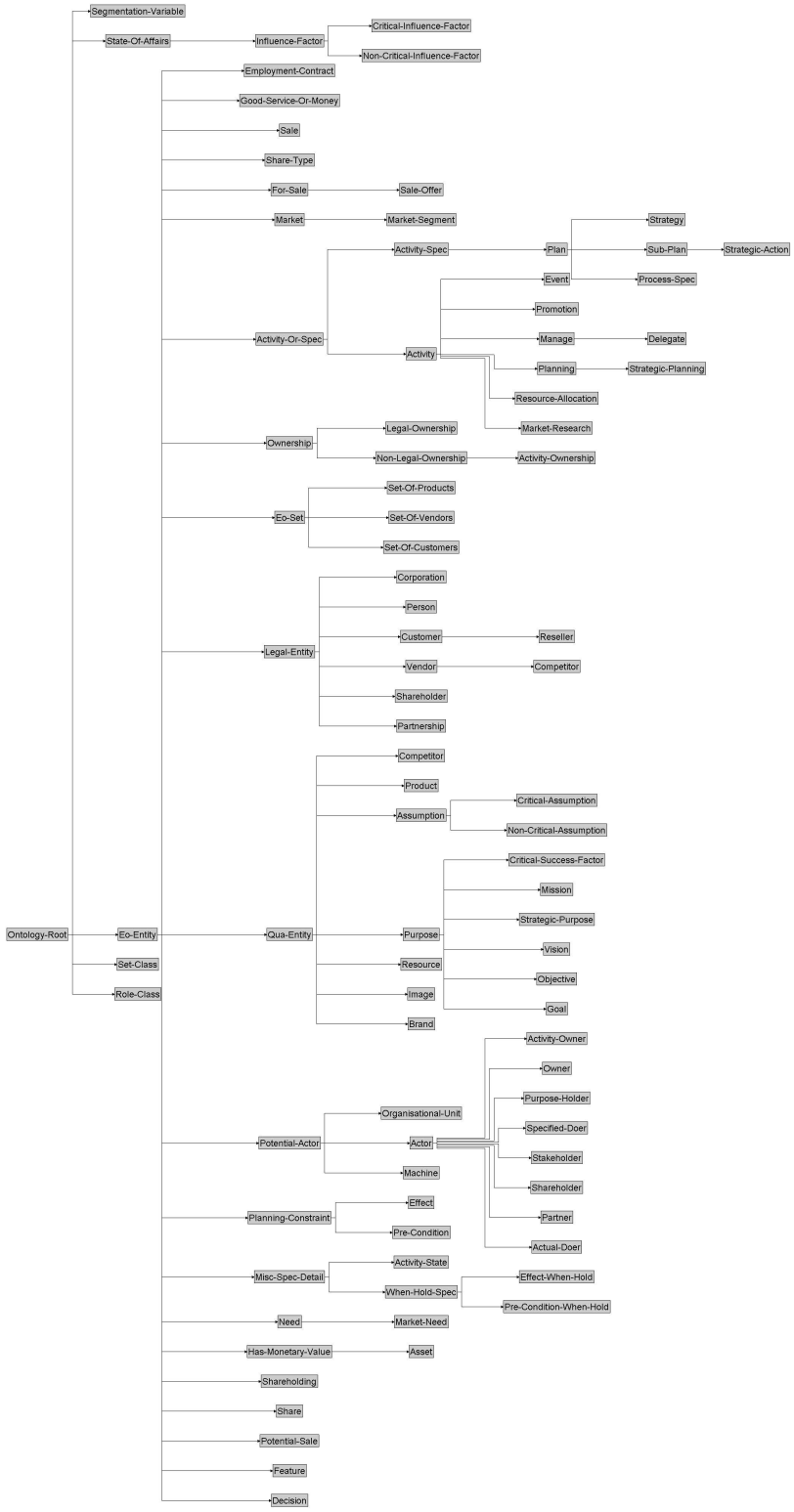
We also observe that the non-dominating functions are not symmetrical about the $y = x$ axis, thus indicating a preference for one dimension over the other. This is due to the asymmetry of the leaf nodes in the data used. These nodes generally contain text, giving them aspect ratios greater than 1.

Figure 2.40(b) shows a plot of the perimeter size measure. In this plot we can see that the area of the layout is a very bad size measure for inclusion layouts. All the trees exhibit very unstable,

area have extreme aspect ratios.

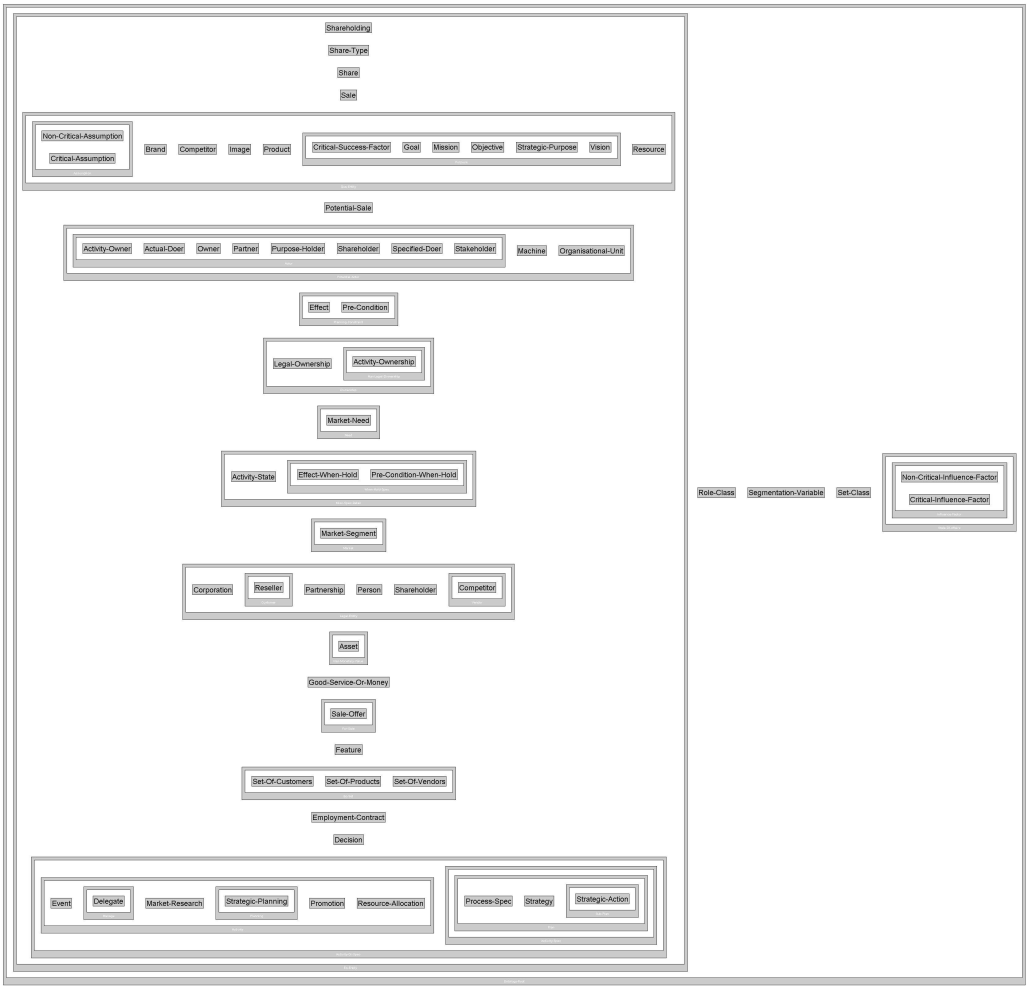
Figure 2.40(c) shows a plot of the perimeter size measure. In this plot, the y coordinate is the perimeter of the layout, that is, $x + y$. This corresponds to the $y = x$ axis in Figure 2.40(a), and so what this shows is equivalent to Figure 2.40(a) "rotated" by 45^o, where the line with $y = 1.0$ and $-1.0 \leq x \leq 0$ in Figure 2.40(c) corresponds to the line with $y = 1.0$ and $0 \leq x \leq 1.0$ in Figure 2.40(a), and similarly $y = 1.0$ and $0 \leq x \leq 1.0$ in Figure 2.40(c) corresponds to $x = 1.0$ and $0 \leq y \leq 1.0$ in Figure 2.40(a). This measure is reasonably good, in that, most of trees have a clear-cut minimum which tends to be near $x = 0$ (square aspect ratio) — particularly the large trees. This is as expected, since this plot is effectively equivalent to Figure 2.40(a).

Figure 2.40(d) shows a plot of the aspect ratio size measure. In this plot all the trees have a very clear cut minimum at $x = 0$, $y = 0$ (with the exception of the trees with only two layouts in their



(a) Node-link Layout Style

Figure 2.39: The Enterprise-Ontology class hierarchy.



(b) Inclusion Layout Style (Minimum Enclosing Square)

Figure 2.39: The Enterprise-Ontology class hierarchy.

non-dominating possible solutions, Man Fishing DBT and Online Shopping High Level DBT). This is not surprising, since the line $x = 0$ corresponds to the line $y = x$ in Figure 2.40(a), the line where layouts are square.

Figure 2.40(e) shows a plot of the minimum enclosing square size measure. In this plot it can be seen that this measure also has a definite minimum at $x = 0$. This is because the minimum enclosing square is minimised when the layout is itself square, which corresponds to the line $x = 0$ ($y = x$ in Figure 2.40(a)), as in the aspect ratio size measure. However, the minimum enclosing square size measure is preferable to the aspect ratio size measure, because it also separates the minimum points in the y axis, whereas the aspect ratio size measure collects them all at $y = 0$. This

is useful, because it means that for two layouts of the same aspect ratio, the one with the smaller edge length is chosen. For algorithms which take dominating rectangles into account, such as the dynamic programming algorithm, this makes no difference, because the rectangle with the larger edge length would be omitted as it is dominating, and the minimum enclosing square size measure is equivalent to the aspect ratio size measure. However, for algorithms which do not use dominating rectangles, such as the greedy algorithm, the minimum enclosing square size measure is expected to be a better size measure. It also allows for easier comparison of the layout quality of different trees, as shown in Figure 2.40(e).

In Table 2.1 we can also see that the number of possible layouts is not monotonically related to the number of nodes. This is because the number of possible layouts, scales with $O(Mn)$ (where M is the sum of the widths of the leaf nodes), as described in Section 2.2.1, rather than with the number of nodes.

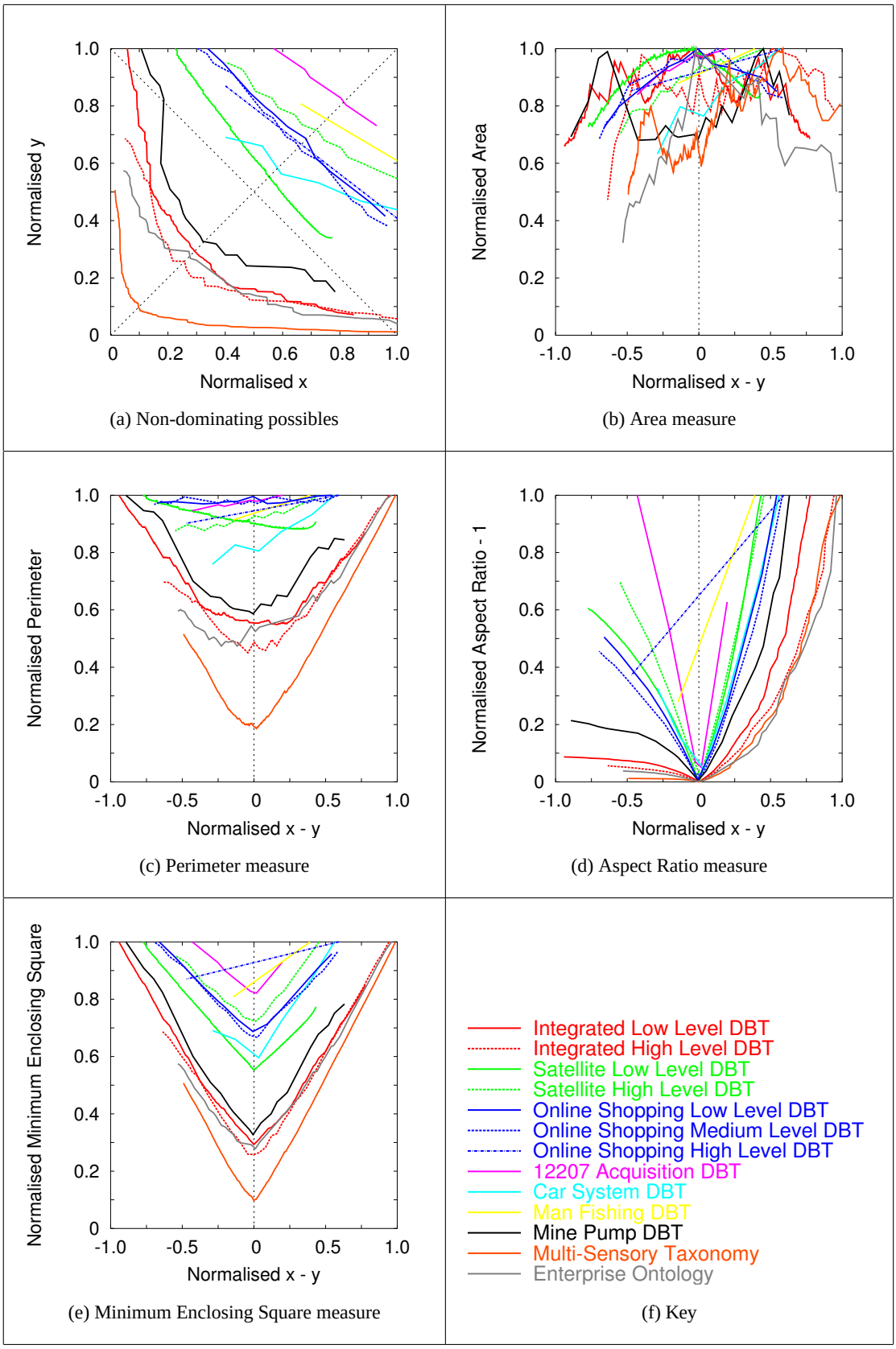
Path compression

As noted earlier, the DBTs have more nodes of degree 1 than the ontologies. Nodes of degree 1 are not well suited to the inclusion layout style, as they result in a single nested rectangle inside another. For these paths of nodes, such nested margins add considerable visual complexity and waste screen space, as can be seen in Figure 2.17.

A better solution for the inclusion layout style, in this case, is to compress each of these paths of nodes into a single representative node. If desired, the representative node can be scaled by an amount proportional to the length of the compressed path. When this occurs at leaf nodes, the resultant inclusion layout approximates that of the original uncompressed tree.

In addition, a visual cue, such as a gradient, may be applied to the representative node, informing the user that some information has been compressed in order to improve the visualisation. However, when the compressed nodes contain text, an application-specific textual summary should be used for the text of the representative node.

Figure 2.41 shows the results of applying path compression to the Mine Pump DBT from Figure 2.17(b). From Figure 2.41(a) it can be seen that the inclusion layout is simpler with paths compressed — although a visual cue would be useful to regather some of the lost information. Figure 2.41(b) shows the same compression, but retaining non-leaf nodes of degree 1 and scaling nodes. No space has been saved, but the node scaling gives some information on the compressed nodes, and may be useful where the size of leaf nodes is of interest. No text summaries have been



(a) Non-dominating possibles

(b) Area measure

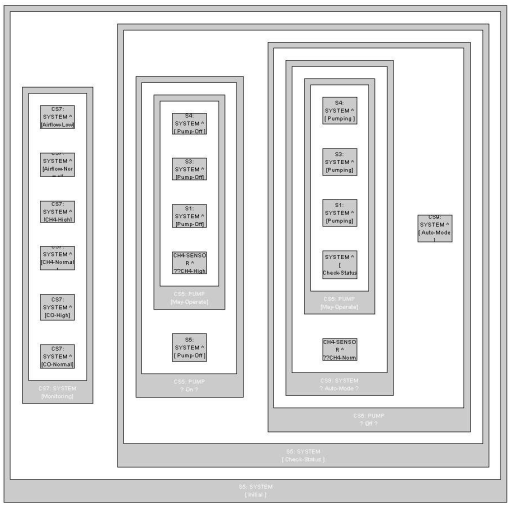
(c) Perimeter measure

(d) Aspect Ratio measure

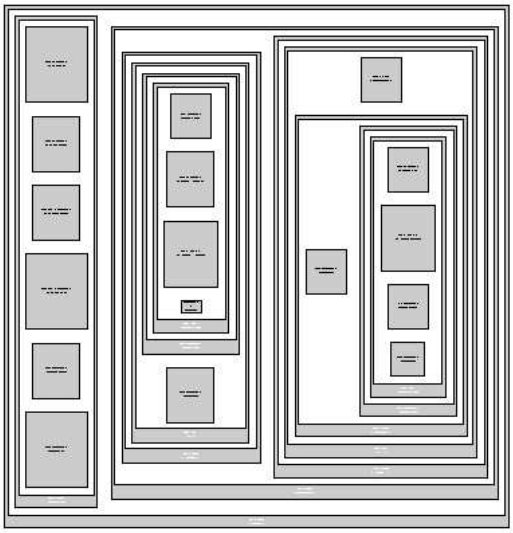
(e) Minimum Enclosing Square measure

(f) Key

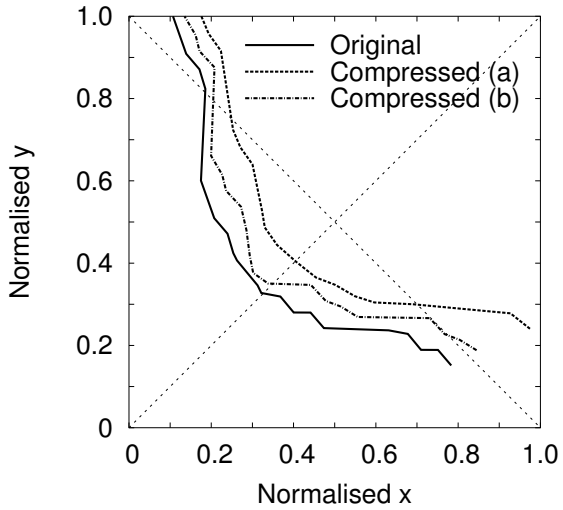
Figure 2.40: Result plots for empirical data.



(a) Inclusion layout style (minimum enclosing square), no node scaling



(b) Inclusion layout style (minimum enclosing square), node scaling



(c) Non-dominating functions

Figure 2.41: Results of applying path compression to the Mine Pump DBT shown in Figure 2.17(b).

generated for the representative nodes. Figure 2.41(c) shows that the non-dominating functions of the compressed trees are only marginally worse than the original, and have the same basic shape.

2.3.2 Conclusion

This section has presented an empirical investigation of the inclusion layout style, based on Design Behaviour Trees from software engineering and ontological class hierarchies derived from artificial intelligence. Several common measures of the size of an inclusion layout have been examined. It was found that area size is a poor measure for the inclusion layout style, and that measures which seek to achieve a desired aspect ratio (in particular, the minimum enclosing rectangle size measure), are more suitable.

CHAPTER 3

Visualisation Model

This chapter presents the visualisation model used in this thesis, and defines *Structural Zooming* in terms of it. Section 3.1 introduces the visualisation model and defines the concept of visual complexity. Section 3.2 describes the abstract model and requirements of Structural Zooming techniques. Section 3.3 presents the methods and techniques in the application of Structural Zooming used in this thesis. Finally, Section 3.4 explains the possible methods for evaluating Structural Zooming, including the approach taken in this work.

3.1 Visual complexity

This section introduces concepts that provide an abstract reference framework for many of the issues faced in large scale visualisation techniques. It is not intended to provide mathematically precise measures for aspects of visualisations; this is supplied for the case of relational information in Section 4.4.

A *visualisation* is a graphical or visual representation of part or all of the data in an underlying *dataset*. The "size" of a visualisation may be measured by two intrinsic properties: the "data content" and the "visual complexity".

The *data content* is a measure of how much of the underlying data is represented in the visualisation, in units appropriate for the type of data being displayed. Data content is quantified by a concrete *data content measure*; for example, the number of nodes displayed in a tree. There are generally many possible data content measures for any given type of data. Choosing the most appropriate is usually an application-specific problem.

The *normalised data content* is the data content as a fraction of the overall dataset size. The traditional problem of determining a useful visual presentation of a set of data has historically involved situations where the normalised data content is 1; for example, nearly all the methods surveyed in

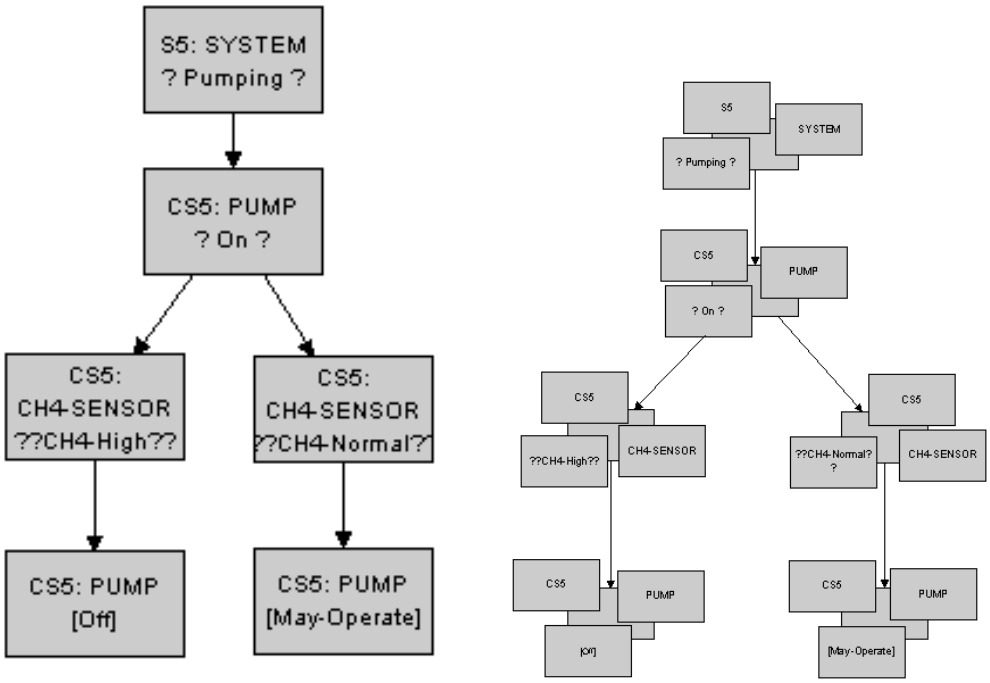
[32] are concerned with computing the static layout of an entire graph. *Large scale* information visualisation, introduced in Section 1.1, consists of techniques that deal with visualisations that have normalised data content less than 1.

Finding an appropriate level of data content for a visualisation is a difficult problem. "Too little", results in too little context, causing inefficient use of screen space. "Too much", results in too much context, causing a lack of "fine detail" and an inability to determine small features in the visualisation.

The *visual content* is a measure of the visual elements or attributes used by the visualisation to present the data. It is quantified by a *visual content measure*, such as Tufte's data-ink [159], the number of graphics primitives used in drawing the visualisation, or the entropy contained in the pixels of the image, as measured by Shannon's information theory [141]. The visual content measure includes all the visual properties used to represent the data. In Nesbitt's taxonomy [107], this includes spatial visual metaphors such as lines and regions, direct visual metaphors such as colour and texture, and temporal visual metaphors such as animation.

The difference between the visual content and the data content may seem subtle, but it is, in fact, an important and rarely considered aspect of information visualisation. Different visualisations of the same data may have different visual contents, which depends on the techniques employed by the visualisation method used. For example, the small node-link tree shown in Figure 3.1(a) has data content of 6 nodes (with 6 text labels) and 5 edges, and visual content of 6 boxes (with 6 text labels) and 5 lines (and 5 arrows). By contrast, Figure 3.1(b) shows the same data, but in a style that is deliberately excessive, requiring 24 partially overlapping boxes (with 18 text labels).

However, the difference between data content and visual content is not merely dependent on the graphical techniques, but also depends heavily on the actual representation of the data itself. For example, consider the dataset shown in Figure 3.2. The "raw" dataset itself is shown using tabulated text representation in Figure 3.2(a). In each of the graphical representations in Figures 3.2(b), 3.2(c) and 3.2(d), each person is represented by a node, and the the 'manager' relationship between them is shown by an edge from the person to their manager. In Figure 3.2(b), the 'location' attribute is represented by supplementing each node with a text annotation. Figure 3.2(c) uses nodes of a different type for the locations (shown in a different colour), and joins a person's node to their location node with an edge of a different type (shown using a dashed line). In Figure 3.2(d), nodes for persons in the same location are grouped together into a cluster node for that location. Exactly the same information (that is, data content) is conveyed in each of these visual representations, but



(a) Classical (b) Excessive

Figure 3.1: A small tree (a section of the Design Behaviour Tree from Figure 2.17(a)), shown using the classical node-link style in (a) and an excessive style in (b). The same data is present in both, but the excessive style has a higher visual complexity.

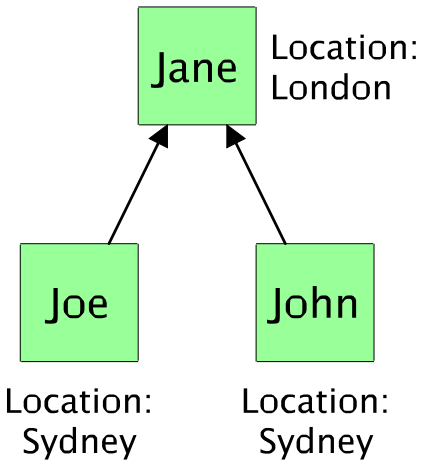
in very different ways, with correspondingly different visual content.

For many visualisations, the visual content is linearly proportional to the data content. The *visual complexity* is the ratio of the visual content to the data content. This is a measure of how succinctly the visualisation captures the data relative to the amount of data being displayed. Whilst data content must be carefully balanced, between being too small and too large, the visual content (and thus also the visual complexity) is always minimised, in order that the clarity and readability of the visualisation may be maximised. The visual complexity has a minimum value of 1, where the visual content is identical to the data content[1]. A visualisation technique exhibits *constant visual complexity* if its visual complexity is independent of the data detail (that is, the visual content is linearly proportional to the data detail). Similarly the technique exhibits *uniform visual complexity* if its visual complexity is constant over all subsets of the data being set out. Both of these properties

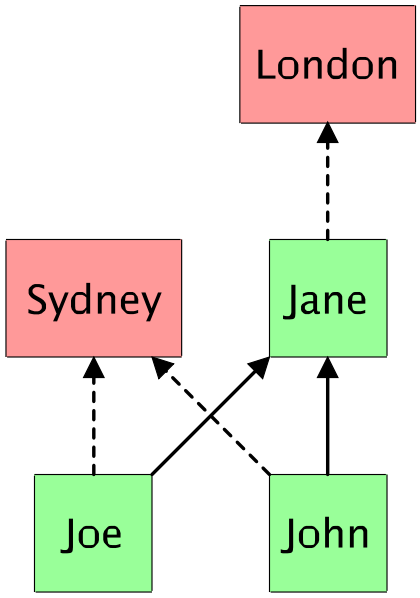
[1]The issue of the data content and visual content having different units is neglected; in practice the visual complexity has a minimum value that can be normalised to 1.

Name	Location	Manager
John	Sydney	Jane
Jane	London	
Joe	Sydney	Jane

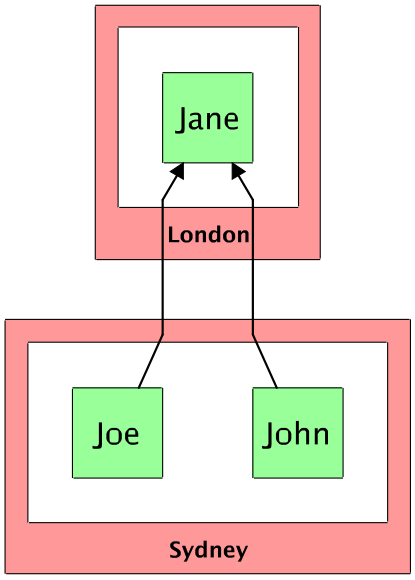
(a) Tabular



(b) Annotations



(c) Mixed types



(d) Clustering

Figure 3.2: An example of three different graphical representations of a dataset that is shown in a “raw” tabular text representation in (a). The location is shown in (b) by textually annotating the nodes, in (c) by using different types of nodes and edges, and in (d) by clustering nodes in the same location.

are highly desirable, therefore it is not surprising that many existing visualisation techniques exhibit one or both of them.

In this model, Tufte's data-ink ratio can be understood as the inverse of a special case of the visual complexity, where the visual content measure is the number of non-background pixels in the visualisation.

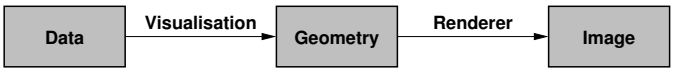
The *data density* is the data content per unit-area on the display device. It has been well studied in the context of statistical scatter plots, with the aim of maintaining a constant data density at all levels of detail [173, 174]. It can be understood as the inverse of the visual complexity in the special case of the visualisation area as the visual content measure. Thus, it does not directly consider the visual content, and assumes it to be linear and monotone.

For a set of data X, the data content is denoted by $D(X)$ and the visual complexity of a given visualisation technique by $V(X)$. The technique has *monotone visual complexity* if and only if $V(X) < V(Y)$ for all $D(X) < D(Y)$. That is, a higher data content visualisation must also have a higher visual complexity, compared to any lower data content visualisation. Here, attention is restricted to the set of visualisation techniques with monotone visual complexity. This simplifies the factors that must be considered when designing Structural Zooming techniques, and ensures that limits placed on the data content result in corresponding limits on the visual complexity.

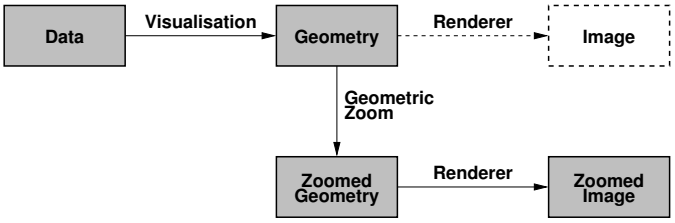
3.2 Structural zooming model

Structural zooming, as its name suggests, is a "zooming" technique that allows the user to examine part of the data in more detail. It is a *data-driven* zooming technique that provides the user with interactive zooming operations which correspond to *logical* operations on the underlying data. In practice, though, the zooming operations do not actually modify the data, rather, they store and modify information about the logical level of detail at which the data is presented. Of particular importance is the *filtering* operation, where data is chosen for display based on some set criteria. This is in contrast to *geometry-driven* zooming techniques, or simply *geometric zooming* techniques, which apply the zooming operations to the geometry produced by the visualisation, rather than to the data itself.

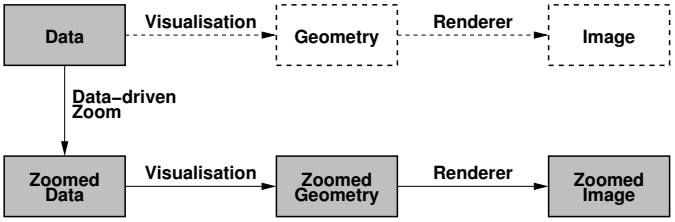
One way of modelling the process of producing a visualisation, is as a pipeline. The canonical *visualisation pipeline* is shown in Figure 3.3(a). It shows that *data* is *visualised* to create *geometry*, that is, a set of *graphical elements* with geometric attributes is constructed from the data. This



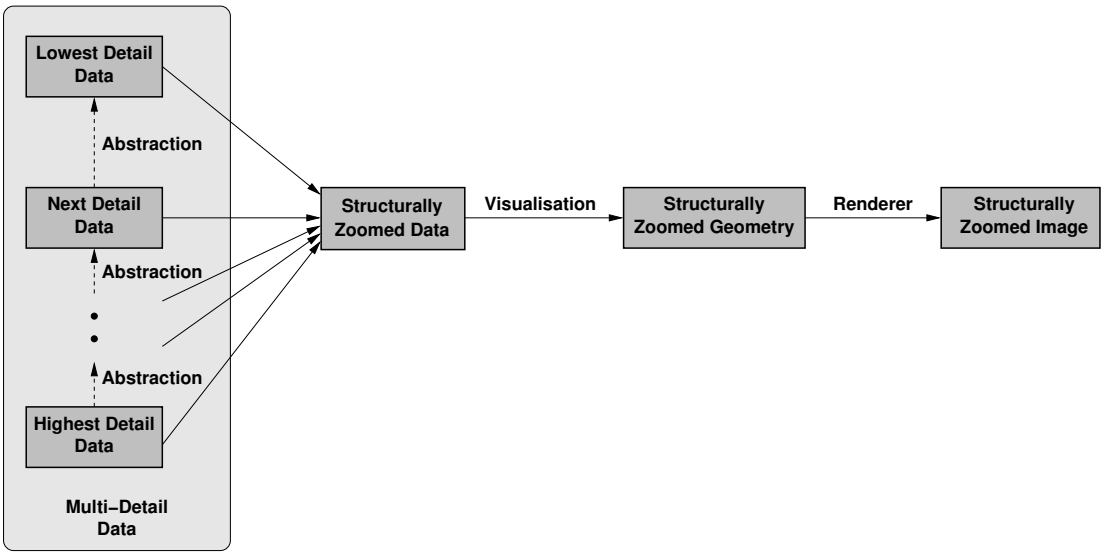
(a) Standard



(b) Geometric zooming



(c) Data-driven zooming



(d) Structural zooming

Figure 3.3: Visualisation pipelines.

geometry is then *rendered* into an *image* that may be viewed by users. Techniques for *geometric zooming* take the geometry produced by the visualisation, and apply a *geometric zoom* operation to it — thus creating a *zoomed geometry*, as shown in Figure 3.3(b). This zoomed geometry typically still includes the majority of the dataset, although the geometric arrangement of the data has been changed so that screen space is disproportionately allocated to the data. Previous work in this area is described in Section 1.2.3.

Data-driven zooming is similar, except that the zoom operation is applied directly to the data, creating a *zoomed data*, as shown in Figure 3.3(c). This zoomed data is then visualised to create the zoomed geometry and image. Existing data-driven zooming techniques, most of which also incorporate some aspect of geometric zooming, is described in Section 1.2.4.

In *Structural Zooming*, shown in Figure 3.3(d), the data itself is separated into a number of *levels of detail*, forming a *multi-detail dataset*. These levels of detail may be intrinsic to the data, or may be determined by successive *abstractions* of a high-detail dataset. Data from the various levels of detail is combined into a *structurally zoomed* (or *mixed-detail*) dataset that seamlessly integrates the data elements of differing levels of detail. This allows the use of existing visualisation and rendering techniques, with the multi-detail dataset.

The aims of Structural Zooming are described in Section 1.3, and form the basis of the operations and properties required by Structural Zooming techniques:

Data-driven Focus + Context: Presentation of the data at different levels of detail, with a main focus region and supporting context region.

Support navigation: Allow the user to change the focus and obtain more detail on selected areas.

Mental map preservation: The value of navigation is greatly reduced if the user is unable to relate elements in the previous view to elements in the current view. Thus, the user's mental map must be preserved during navigation.

Avoid geometric distortion: Geometric distortion may not be used in the visualisation. Data-driven methods of allowing the user to 'zoom' into specific areas of the data must be used.

Constant visual complexity: To assist users' understanding of the visualisation and avoid overwhelming them, the visual complexity should be maintained

at an appropriate and approximately constant level of visual complexity during navigation. It is assumed that the visual complexity is uniform over the visualisation, that is, the visual complexity is only considered "globally" over the present visualisation, and not at a "local" level within the visualisation.

Good quality layout: The aesthetically pleasing visual layout and presentation of data possible in static visualisations, should also be present in Structural Zooming.

The interaction style used for navigation in Structural Zooming is assumed to be a *direct-walk*. This consists of a sequence of discrete operations, where each operation selects a data element in the display — usually by clicking on its visual representation with the mouse. This type of navigation has been shown to be efficient, in terms of maximising the information available to the user, compared to the time spent searching [26, 60].

The short-term memory of the user cannot be neglected in the consideration of the navigation technique of any large scale information visualisation technique. Short term memory allows the user to remember a small number of aspects of previous visualisations. However, the limited nature of short term memory restricts both the number of objects that a user can recall, and the length of time for which they can be recalled (measured as the number of transitions). Specifically, the recollection time is inversely related to the number of objects that are displayed (that is, the more objects shown, the sooner the user will "forget" them). For a typical visualisation, the user's history is likely to be limited to, at most two or three previous visualisations. This means that individual static presentations should not contain too much information, as the user is unable to retain much of it once it has left the view. The converse of this, is that it is sometimes possible to deliberately display slightly less information (particularly context information), in the knowledge that users can retain a higher proportion of it when the visualisation changes. Both these aspects are enhanced by the use of animated transitions between visualisations. This is because the animation allows elements of the previous visualisation to be "converted" or "transformed", into elements of the current visualisation that are likely to be reinforced in short term memory, rather than lost or rearranged within it. These considerations tend to support the use of an approximately constant visual complexity.

In addition, data-driven Focus + Context zooming techniques have the advantage of emphasising the data itself during navigation. By contrast, geometric distortion may tend to emphasise particular visual and geometric characteristics of the presentation of the data.

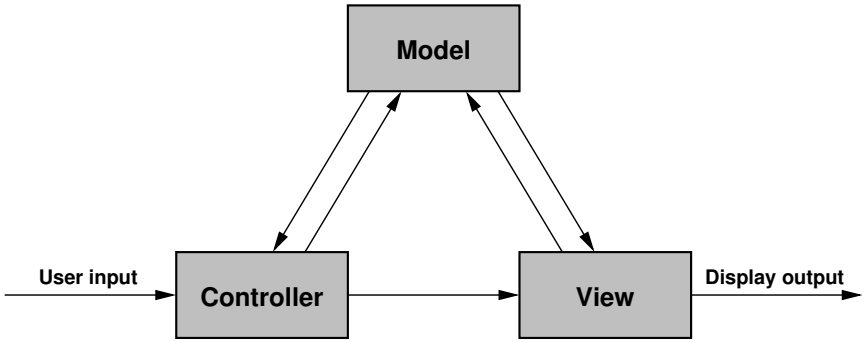


Figure 3.4: The Model-View-Controller (MVC) architecture.

Structural Zooming is most similar to the geometric and non-geometric Focus + Context techniques presented in Sections 1.2.3 and 1.2.4. The principal difference is the use of data-driven zooming, with an approximately constant level of visual complexity. This avoids the problems associated with Graphical Fisheye Views, presented in Section 1.3, that is to say, low focus data density, high context data density, poor spatial properties and difficult multiple foci.

The concept of a multi-detail dataset is similar to the *multi-resolution meshes* found in computer graphics [57, 74]. A multi-resolution mesh is a collection of several computational meshes of a geometric model. Each mesh is stored at a different resolution (that is, a different level of detail) in a hierarchical fashion that allows rapid access to any part of the mesh at any resolution. This is used to increase the performance of real-time graphics systems by presenting models near the viewer, at high detail, and distant models at low detail, with seamless transitions between levels of detail. Another example of multi-detail datasets may be found in Geographic Information Systems (GIS). Such systems contain *features* stored in spatial data structures. A feature is a spatial object that occupies a region of the landscape in the system, such as towns, roads, houses, rivers, fields, and so on. In these systems, it is common for features to be represented with different levels of detail depending on the zoom level. For example, when zoomed out to view a large area such as a state, each town could be represented as a dot. Upon zooming in closer, roads and houses within towns are shown. This idea can be observed in Figure 1.3, for example, the difference in the representation of Alfred Street in Figures 1.3(b) and 1.3(c). Structural zooming may be thought of as an extension to this model, where the features in a visualisation may be at different zoom levels, and thus displayed at different levels of detail.

The *Model-View-Controller* (*MVC*) paradigm is a commonly found architecture for modelling and implementing interactive graphical systems. It was introduced in the Smalltalk-80 system [63,

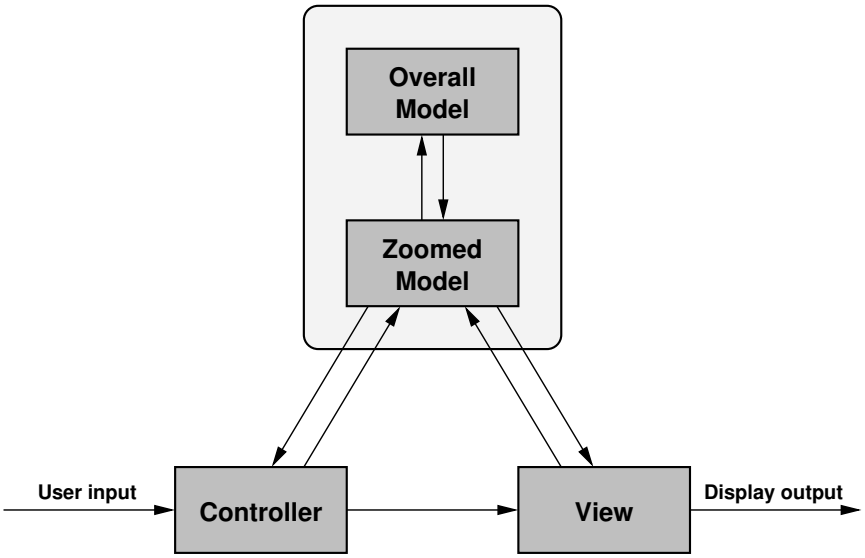


Figure 3.5: The MVC architecture of Structural Zooming.

64], although literature did not appear until considerably later [85, 86]. As its name suggests, the MVC is composed of three components, the "Model", the "View" and the "Controller", which communicate as shown in Figure 3.4. The *Model* is the abstract representation of the underlying data. The *View* is the visual or graphical representation of the Model which is presented to the user. The *Controller* is the user interface aspect, that is, how the user is able to input adjustments of the Model and View.

In Structural Zooming, the Model actually consists of two parts, shown in Figure 3.5. It contains the entire dataset being visualised, but, more importantly, it contains the structurally zoomed version of the data. Of course, the structurally zoomed data need not be stored independently; it may be possible to augment the main dataset with supplementary information, thus allowing the structurally zoomed sub-dataset to be easily and efficiently extracted. Changing or updating the main dataset may be supported by the Model, but this is not essential. The View is the visual representation of the structurally zoomed data that is presented to the user. The Controller deals with how users interact with the visualisation, more specifically, how they are able to control the level of detail, and how they are able to navigate and move the focus region throughout the dataset.

3.3 Structural zooming techniques

Structural zooming systems limit the level of visual complexity in the visualisation to be below a constant value. This allows them to maintain an approximately constant level of visual complexity, where conventional visualisation of the data would ordinarily require much higher visual complexity. In this context, distortion techniques, such as geometric zooming, generally tend to have higher visual complexity, caused by "packing" large amounts of detail into a relatively small context region. Rather than drawing all the context data in a compressed and distorted fashion, a better approach taken by Structural Zooming is to "summarise" the context data, giving lower detail and thus lower visual complexity.

In order to be useful, Structural Zooming must provide operations supporting the user in directing the visualisation of the data. Since a Structural Zooming system is an interactive Focus + Context system, this means providing *user navigation operations*, or simply *user operations*, that allow the user to change the focus region and control which data is included in the visualisation. Thus, they are *detail-increasing operations*, which therefore, by necessity, also increase the visual complexity. It is the role of the Structural Zooming system to maintain the approximately constant level of visual complexity, which means that the system must reduce the detail in response to the user's increases. In particular, for every type of detail-increasing operation available to the user, a corresponding inverse *detail-reducing operation* is available to the system.

When the user increases the detail by performing an operation (the *stimulus*), the system must determine:

- The *detail reducing condition*

  If the data content has become too high, as a result of the stimulus operation, the detail reducing condition is true. The simplest way of determining this, is if the data content measure has risen above some pre-determined *maximum detail threshold* value.

- The *response*

  If the detail reducing condition is true, the response is how the system is to return the detail to acceptable levels. This must be achieved without disturbing the user's navigation or investigation process. Notably, the response cannot include the inverse operation of the stimulus.

It is also possible to allow the user access to perform the detail-decreasing operations. The user

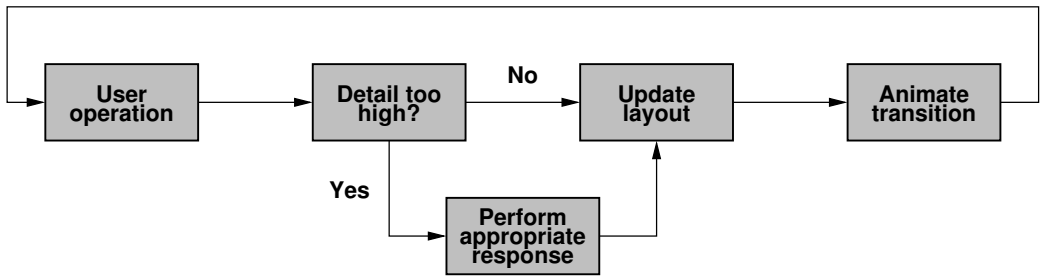


Figure 3.6: A flowchart showing the central interaction loop of Structural Zooming.

always has a better idea of the overall goal than the system, and thus may choose to anticipate a better "response" to a stimulus yet to be performed. However, since the main task of the user is searching for information in the visualisation, the provision of detail-decreasing user operations is completely optional. The system does not respond to any detail-decreasing operations performed by the user. Symmetry would have the system increase the detail, but clearly, how this should be done is not a choice the system is best equipped to make.

The *layout* or *presentation* of the visualisation must be updated as the user changes the view. This is to ensure that the visualisation quality is preserved, in such terms as good aesthetic properties, clarity and readability. This is usually achieved with an automatic layout algorithm for the type of data involved. However, it is important to ensure that the updated layout is sufficiently similar to the previous layout, to prevent the user from becoming "lost" in the visualisation. The *mental map* is the user's internalised representation of the visualisation. The preservation of the mental map, as characterised by certain visualisation properties, is important for users to maintain their orientation and understanding of the data [41, 43, 101, 149]. In addition, all on-screen transitions should be smoothly animated as this is an effective strategy for mental map preservation [175]. However, simple animation techniques, such as linear interpolation, may not be sufficient [54], meaning that specific animation strategies for each type of data and each operation may be required. It is preferable for the animation of an inverse operation to be the time-reversed animation of the original operation, but this is not essential.

The stages of Structural Zooming are shown diagrammatically as a flowchart in Figure 3.6. It demonstrates that user operations are followed by a detail evaluation. If the detail reducing condition is true, then the system determines suitable responses for returning the detail to an acceptable level. This is followed by the recomputation of the layout and an animated transition from the previous layout to the new layout.

Animations in Structural Zooming are assumed to be "independent", that is, the animations of two disjoint data elements are also visually disjoint. This allows animations to be *composed* together. This is necessary when the system needs to perform more than one response operation, or when the response involves a large number of data elements. Animations may be composed consecutively or concurrently.

Consecutive animation, is when each component animation is performed serially. This introduces the issue of the order in which the component animations should be effected. There is also the problem of signalling the end of the overall animation to the user without resulting in hestitation or confusion. The user may be unsure if a subsequent animation will commence after each individual animation, causing either an unnecessary wait or a further navigational attempt before the overall animation has completed. This may be alleviated by increasing the frame rate of each component animation, so that the overall animation always takes some fixed amount of time, however, this is infeasible when there are many component animations.

Concurrent animation, is when all the component animations are performed together, in parallel. It has the problem of potentially confusing the user if there are many component animations, causing large amounts of movement. Research in perceptual psychology has shown that humans are able to track about four independant motions in parallel with little cognitive effort [126]. Thus, concurrent animation is likely to be acceptable when the motion is below this limit, and unacceptable otherwise.

This suggests a hybrid animation strategy, where the individual animations are partitioned into groups of four. The animations within each group are animated concurrently, while the set of "group" animations are consecutively animated. However, the partitioning and ordering problems, mean that the design of such a strategy is outside the scope of this thesis, which takes the simple approach of performing all animations concurrently.

3.4 Evaluation methodology

There are several possibilities for the evaluation of interactive Focus + Context techniques, such as Structural Zooming. The most common of these, including the method used by this thesis, are now reviewed.

User studies is the name of a method commonly found in the fields of user interfaces and perceptual psychology. Such studies involve conducting a controlled scientific experiment with a

collection of humans performing a particular set of tasks. The goal is to provide support for, or against, particular hypotheses. In the case of interactive visualisation systems, the hypothesis is usually about the user's ability to perform some operation or task in the system being tested. The experiment must be very carefully designed to ensure that only one variable is changed, and there is no bias from other controlled or uncontrolled factors. In addition, user studies are logistically quite involved, and the required statistical analysis of results can be difficult.

Mathematical analysis can be used in various situations to prove that benefit is obtained by the use of particular techniques. However, this usually relies on earlier results which have been obtained using other methods, such as user studies, or psychological and physiological analysis of the senses. For example, suppose a result has been obtained from user studies which states that users are successfully able to determine connected nodes in a graph if edges contain at most three bends. It would then be desirable, to prove mathematically, that a particular graph drawing algorithm always produces edges with at most three bends. As such, mathematical analysis usually supplements other evaluation work.

Empirical measures involve experimentally running a technique with a large corpus of test data, and evaluating the performance against a set of predefined quantitative *quality measures*. These are quantitative measures of qualitative aspects of the visualisation and navigation system, similar to the graph drawing *aesthetics* used in that field [125]. The measures are objective, but there is an underlying assumption that they adequately reflect the actual quality of the system, as perceived by users. In many cases, existing literature in the field of information visualisation and perceptual psychology, is used to justify the choice and definitions of measures. Most quality measures are relatively simple and draw on fundamental principles of perception and design, and are thus acceptable to many researchers. However, fully validating an empirical quality measure, in terms of how it applies to users, usually requires an appropriately designed user-study. The selection of test datasets for the corpus is important. Generally speaking, a corpus is better when it has a large number of datasets with varying statistical properties, and the datasets are derived from "real-world" applications, rather than being artificially constructed. These criteria are required to support the application of the results of the empirical study to actual situations of use.

This thesis adopts a ***hybrid*** evaluation method, primarily based on empirical measures, but also incorporating some aspects of user-studies and mathematical analysis. Since this work evaluates a navigation technique in addition to an information visualisation strategy, a corpus of datasets alone is not sufficient. We additionally require a corpus of *navigation data* for each test dataset. This

navigation data is a description of a *navigation path* through each dataset, that is, a sequence of Structural Zooming navigation operations. This allows the Structural Zooming technique to be run several times over the same set of data and operations, each time with different parameters or settings. Although this assumes that the same navigation path would have been taken in each scenario, it allows a large number of parameters to be widely varied and subsequently directly compared. It is possible to obtain a user navigation path for each combination of parameters, however, if this is very large then it can be taxing on the user. Additionally, significant learning effects may need to be taken into account as the user learns the dataset during their lengthy exposure to it.

The approach for obtaining navigation data is similar to that used if a user study were being performed, but without much of the complexity required by a full user study. Navigation data is obtained, in the first instance, by recording the actions of users exploring each dataset in the corpus using the Structural Zooming technique. This allows the entire sequence of navigation events, known as a *logfile*, to be easily and repeatedly "replayed", each time with different parameters. It has the advantage that the navigation is being performed not only on actual corpus test data, by actual users, but it is using the Structural Zooming technique in question, and is thus representative of the kinds of navigation performed by users in such systems.

Although it is relatively easy to record user navigation actions after instructing users to "explore" the contents of the dataset, each navigation path still requires the active participation of at least one user. When the corpus contains many files, each of which is very large, this can become a burden. A better approach is to devise *navigation strategies* that are capable of automatically generating navigation paths. In an experiment, a navigation strategy can be used to generate navigation paths to supplement those obtained from logfiles. Navigation strategies are introspectively deduced by the experiment designer, based on observation of the exploration strategies exhibited by users. This means that they are only useful for comparing the parameters of particular Structural Zooming methods, and not for evaluating Structural Zooming in terms of a particular application or task, something that would typically require a user-study. Navigation strategies also give the experiment designer more flexible control of the statistical properties of the navigation data, which may be useful when analysing the results.

Thus, the hybrid evaluation methodology adopted by this thesis is primarily an empirical study, based on the measures defined in Section 4.4. Navigation paths for the corpus are obtained from both logfiles and navigation strategies, which are defined in Section 4.5. Chapters 5, 6 and 7 each present an empirical study within a particular application domain.

CHAPTER 4

Relational Data

This chapter describes the application of Structural Zooming (described in Chapter 3), to the visualisation of relational data, namely, trees and clustered graphs. In particular, the multi-detail structure takes the form of a tree, and forms a natural part of the data. For relational datasets that do not have an inherent multi-detail structure, one may be deduced by using *abstractions* (Section 3.2), such as a recursive graph clustering algorithm.

As described in Section 2.1.2, clustered graphs are considered to be an extension of inclusion style trees. As such, Structural Zooming is first applied to trees (both node-link and inclusion styles) in Sections 4.1 and 4.2, and subsequently applied to clustered graphs in Section 4.3 by considering the addition of node-link edges to the inclusion tree case. Section 4.4 presents the empirical evaluation measures for Structural Zooming of relational data. Section 4.5 presents the navigation strategies for Structural Zooming of relational data.

4.1 H-V layout

This section presents an application of Structural Zooming to tip-over and inclusion trees using the optimal h-v layout algorithm presented in Section 2.2.1.

4.1.1 Structural zooming technique

The concept of detail in a tree, corresponds very naturally to the *levels* in a tree, shown in Figure 4.1. The root node alone represents the entire tree at the lowest level of detail, intermediate non-leaf nodes represent their sub-trees at intermediate levels of detail, and leaf nodes represent only themselves at the highest level of detail.

In this multi-detail model of trees, the structurally zoomed version of the original tree is de-

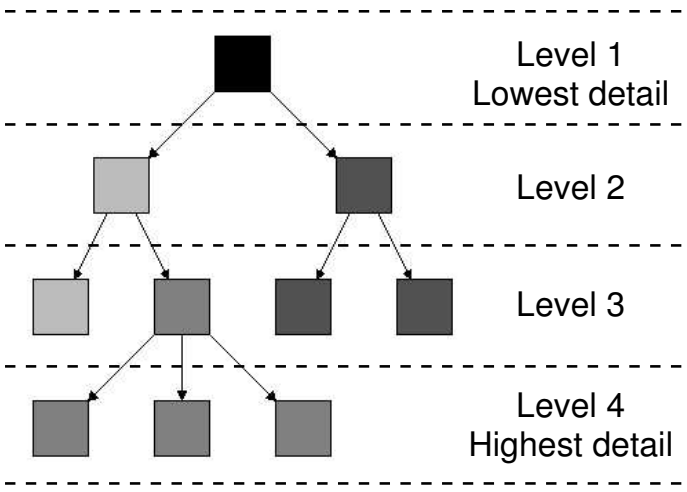


Figure 4.1: Levels of detail in a tree.

termined by an *abridgement* of the tree. Described in Section 2.1.1, an abridgement shows only the ancestors of the nodes of an antichain, which is a "horizontal cut" through the tree. This immediately suggests the detail-increasing user operation of *expanding* (or *opening*) nodes that are collapsed in the abridgement. This adds the children of the node to the abridgement, displaying them and allowing them to be further expanded, if any are collapsed. The inverse detail-decreasing system operation is to *collapse* (or *close*) an expanded non-leaf node in the abridgement. This removes all its descendants from the abridgement, changing the non-leaf node into a collapsed node.

Thus, a typical navigation operation proceeds as follows. The user requests the expansion of a node, typically by clicking on it with the mouse. The system then determines the increase in detail that would occur as a result of this, and compares it against the maximum detail threshold (described below). If the detail has increased by an unacceptable amount, then the system chooses one or more nodes to collapse in order to maintain the detail below the maximum detail threshold.

In order to determine which nodes are to be collapsed, the system maintains a *least recently used* (*LRU*) queue of expanded nodes, known as the *expanded node queue*. When a node is expanded, it is added to the end of the queue, and is followed by each of its ancestors, in order from the expanded node up to the root. The ancestors of the node must have been expanded previously, and so are present earlier in the queue. Each node must appear at most only once in the queue, so in fact, the ancestors are moved from their previous positions in the queue to the end. Now, when the system is required to collapse one or more nodes, it simply traverses the expanded node queue from the head, collapsing nodes until the maximum detail threshold criteria is met. The specific order

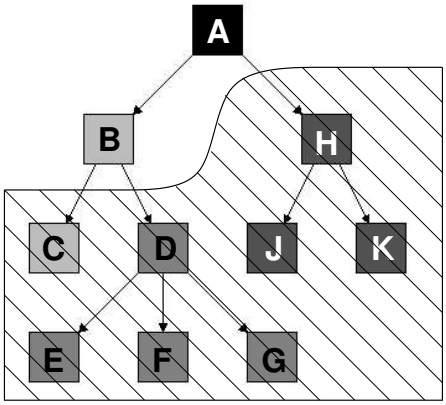
of the queue guarantees that nodes are collapsed starting with the deepest expanded nodes. This prevents the undesirable situation where a node is collapsed even though it has descendants that have been expanded more recently. In addition, none of the ancestors of the node being expanded should be collapsed in response to the expansion. This is easily achieved by stopping the expanded node queue traversal when the node being expanded is found. Figure 4.2 shows the effect of an example node expansion on the expanded node queue.

In fact, this LRU-based strategy is one of several ways of determining the nodes to be collapsed. The most notable alternative would be to use Furnas's fisheye Degree of Interest (DOI) measure, to collapse nodes with the lowest DOI value. Other strategies may use the graph theoretic or Euclidean distance metric between nodes. In addition, in all such node collapsing strategies, the implementation of multiple focus regions can be achieved by allowing the *pinning* of nodes by the user, so that pinned nodes are never chosen to be automatically collapsed.

However, node expanding and collapsing alone are not sufficient. The situation may arise where the expanded node queue only consists of one node and its ancestors. This occurs when the user is "drilling down", repeatedly expanding nodes inside the previously expanded node. In this case, when the maximum detail threshold is exceeded, it is not possible to collapse any nodes. This means that in deep trees, the data content measure can continue to increase as the tree is explored more deeply. Since the abridgement is drawn at an appropriate size for filling the display, yet the layout size continues to grow (since the size of the abridged tree is growing), the resolution of the abridgement decreases. Figure 4.3 shows an example of this. Avoiding this problem is the motivation for "level zooming" in Structural Zooming of trees.

Level zooming operates by distinguishing one non-leaf node to be the *pseudo-root* node, r_p. The pseudo-root is initially the root node, $r_p = r$. The abridgement is drawn such that the region of the pseudo-root $R(r_p)$ fills the screen, rather than the region of the root $R(r)$. The pseudo-root node must always be an ancestor of the most recently expanded node. This gives the system another detail-reducing operation, *level zooming in* (or *zooming in a level*), that it can perform if no nodes can be collapsed. This involves making the new pseudo-root node be the child of the current pseudo-root node which is also an ancestor of the most recently expanded node and hiding the siblings in the process. This can be repeated as many times as is necessary in order to maintain the maximum detail threshold.

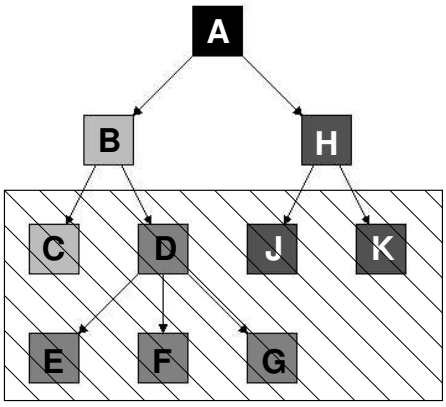
To allow the user to access the parts of the tree hidden by level zooming, the additional user operation of *level zooming out* (or *zooming out a level*) must be provided. This simply performs the



B	A

Queue

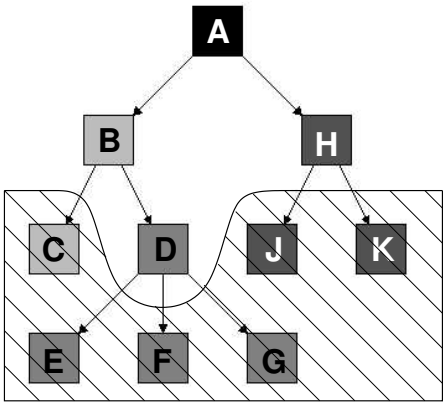
(a) Initial



B	H	A

Queue

(b) Expand H



H	D	B	A

Queue

(c) Expand D

Figure 4.2: An example of how the expanded node queue updates after successive node expansion operations. The non-hatched regions indicate the visible expanded nodes of the tree. The queue is ordered such that left-to-right corresponds to head-to-tail.

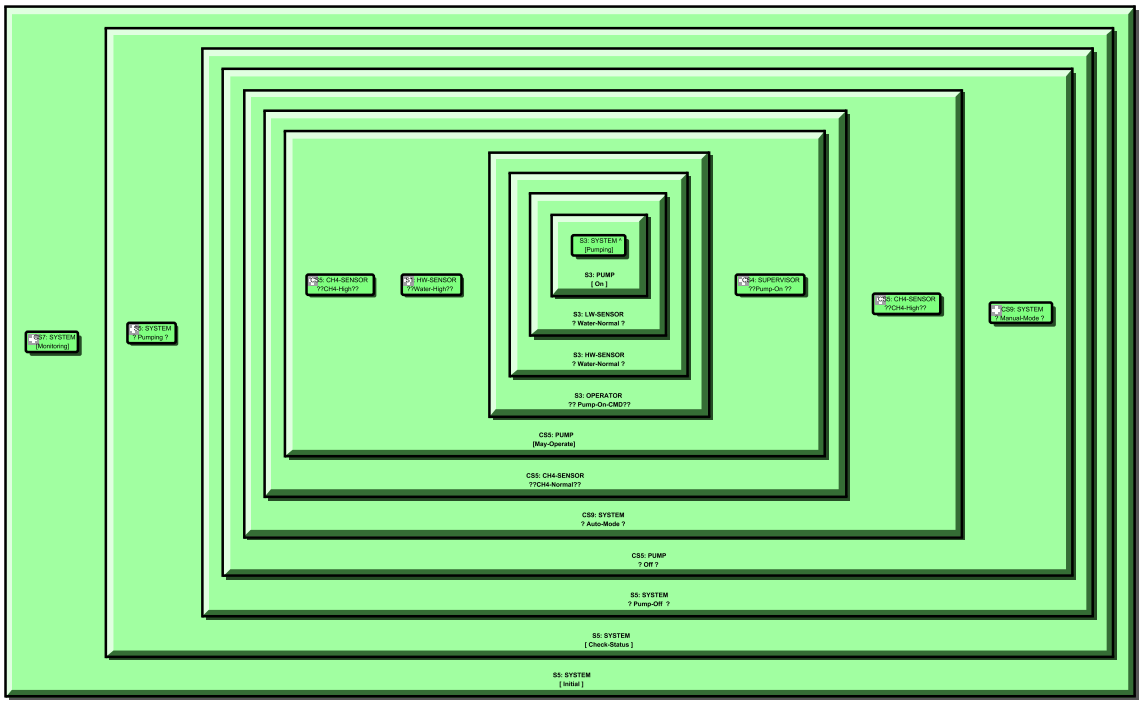


Figure 4.3: An example of a deep tree that has been explored by “drilling down”.

inverse operation, making the new pseudo-root the parent of the current pseudo-root node. Since this operation causes the siblings of the current pseudo-root node to become visible, it is a detail-increasing operation. This means that the system may need to reduce the detail in response to it. Obviously, level zooming in is not useful as a response, since it would exactly negate the requested level zoom out. Thus, the system can only respond to a level zoom out by collapsing nodes. (This is always possible, since the nodes can always be collapsed upwards as close to the root node as necessary.) Figure 4.4 illustrates the level zooming operations.

To summarise, the detail-increasing operations available to the user are:

- Expanding a collapsed node
- Level zooming out

The detail-decreasing operations available to the system are:

- Collapsing an expanded node
- Level zooming in

The response of the system to level zooming out:

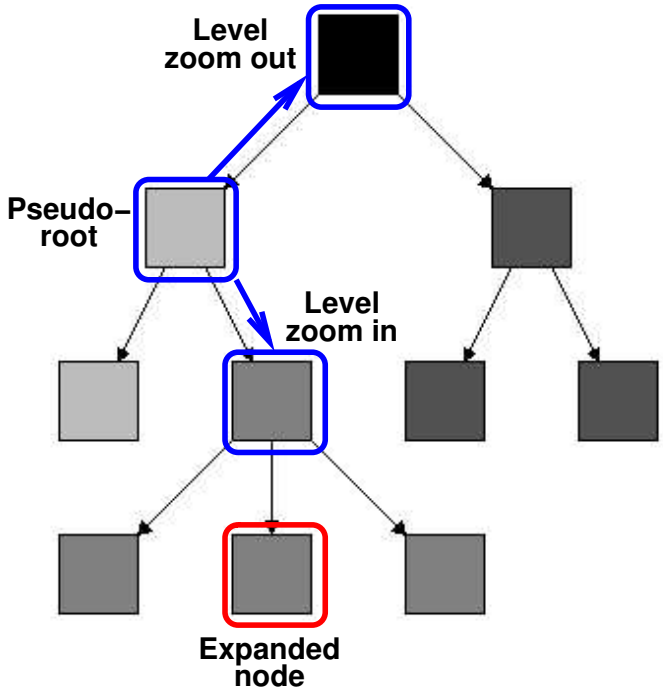


Figure 4.4: Level zooming by changing the pseudo-root node.

- Cannot include level zooming in
- May include the collapsing of any nodes

The response of the system to the expanding of a node:

- Cannot include the collapsing of that node or any of its ancestors
- May include collapsing of any other node
- May include level zooming in one or more levels

One of the aims of Structural Zooming is to maintain good visualisation aesthetics. To ensure that this is achieved, the layout algorithm is run after each operation, recomputing the placement of the nodes. The transition between the old and new layouts is animated, as described below in Section 4.1.2. Only the layout for the subtree rooted at the pseudo-root node is updated, while the layout for the off-screen part of the tree (that is, the part of the tree that does not have the pseudo-root node as an ancestor) does not change. This helps to preserve the user's mental map, as hidden parts of the visualisation do not change appearance whilst out of view. However, there are two possibilities for updating the layout inside the pseudo-root node:

1. Recompute the layout for the entire tree, but only update the positions of descendants of the pseudo-root.

2. Recompute the layout as though the pseudo-root is the overall root of the tree.

Put another way, although only the descendants of the pseudo-root are updated, the layout is recomputed using either the actual root or the pseudo-root of the tree as the root of the layout.

Using the root has the major disadvantage that the quality of the layout of the pseudo-root may be poor when taken on its own, as the layout algorithm optimises the layout of the overall tree. For example, the region of the pseudo-root may have an extreme aspect ratio (compared to the desired aspect ratio), because the h-v layout algorithm has optimised the layout so that the overall tree has a good aspect ratio. In addition, the size of the overall abridgement may be quite large, whereas the size of the on-screen subtree is known to be managable. Thus, performing the layout for the entire abridgement may be too computationally expensive, despite the use of efficient algorithms.

Using the pseudo-root has the disadvantage that the resulting layout may be quite different from previous layouts, resulting in large layout changes when zooming in. However, the animation used helps to alleviate this problem somewhat, and the advantage of having an optimal layout far outweighs this disadvantage. It also makes sense to use the pseudo-root as the root of the layout computation as it is the root of what the user can see. For these reasons our system uses the pseudo-root as the root of the layout computations.

The final consideration for Structural Zooming of h-v trees is the choice of data content measure. Several simple data content measures are possible:

Number of nodes (NN): This performs poorly for deep trees, where much of the maximum detail threshold is occupied by non-leaf nodes that are simply the path taken to the area the user is interested in. As such, this data content measure is better for trees that are not very deep, and trees where the values of all non-leaf nodes are equally as important as the leaf nodes.

Number of leaf and collapsed nodes (NC): This avoids some of the problems in using the number of nodes, but has the problem that nodes of large degree can use much of the maximum detail threshold. This detail measure is better for deeper trees that have limited degree, and where the values of the leaf nodes are more important than the values of expanded non-leaf nodes.

Number of expanded nodes (NE): This is useful for trees that have larger degrees. When a node of very large degree is expanded, it is possible that it may have more children than the maximum detail threshold, forcing all other expanded nodes to be collapsed. The reasoning behind counting only the expanded nodes, is that they are the nodes the user has directly shown interest in (through expanding them). However, this measure is less useful for deep trees and trees with small degrees, where many nodes must be expanded in order to reach a small number of leaf nodes.

Since the choice of data content measure depends on the characteristics of the data, the empirical investigations each examine different detail measures, as appropriate. The investigation in Chapter 5 uses the number of leaf and collapsed nodes, while Chapters 6 and 7 use the number of leaf and collapsed nodes and the number of expanded nodes.

The concepts presented in this section are now illustrated with an image gallery of three example operations.

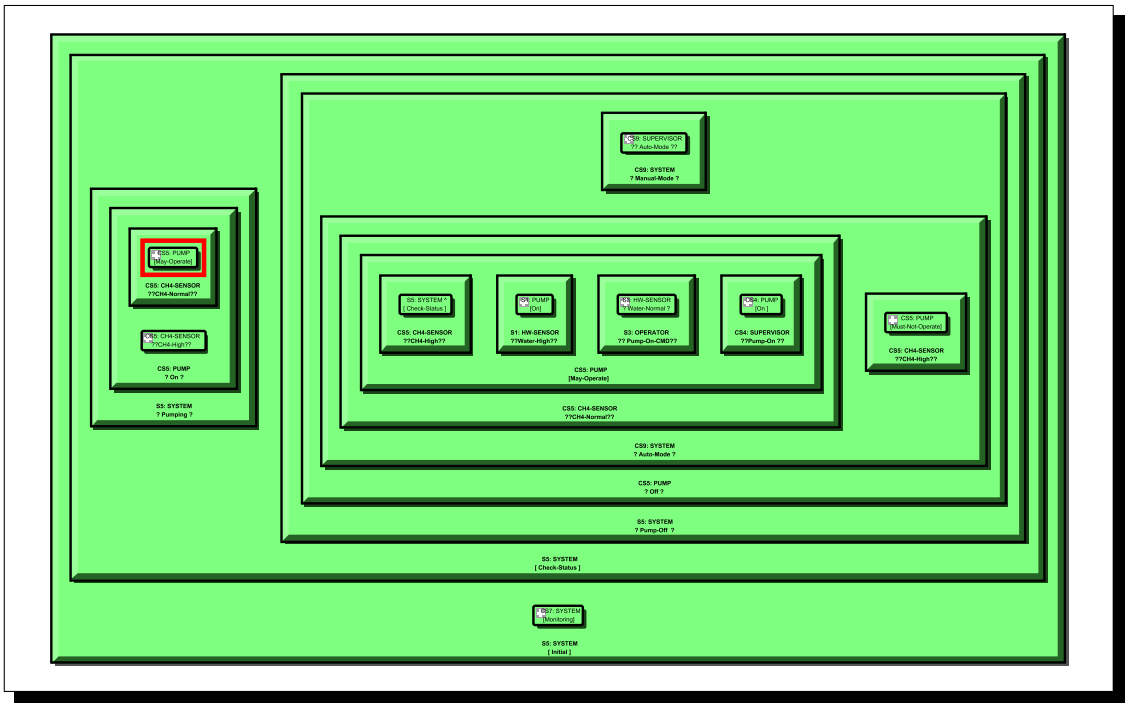
Figure 4.5 shows the effect of expanding the upper of the two left-most collapsed nodes. It is observed that the large expanded node originally occupying most of the display has had many of its descendants collapsed, and now occupies a smaller area on the right. The root node has changed from vertical to horizontal layout, and the layout of several other nodes has also changed. The two displays have similar sizes, amounts of wasted space and numbers of nodes.

Figure 4.6 shows the effect of expanding the bottom-most collapsed node. In this case, the node expands to reveal six children, laid out vertically on the left of the display. However, the remaining expanded nodes have collapsed considerably in order to make sufficient space for this. The right expanded node has collapsed completely, whilst the children of the left one have collapsed. Again, the displays are similar in detail and size.

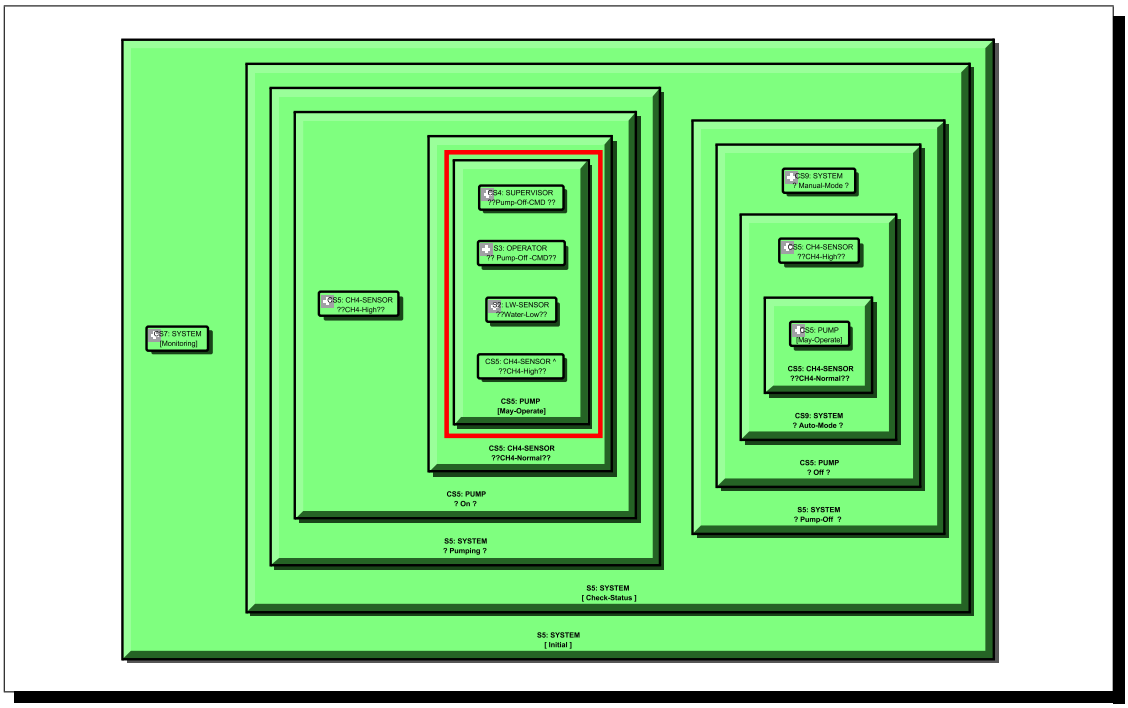
Figure 4.7 shows the effect of level zooming. It shows how the visualization in Figure 4.3 appears when level zooming is enabled. It is seen that there are less on-screen nodes, and that those present are larger, more easily readable and closer to the central node which was expanded.

4.1.2 Animation

The example Figures 4.5, 4.6 and 4.7, illustrate how these transitions would appear to the user in the absence of any animation. Considerable effort is required to determine the changes that have taken

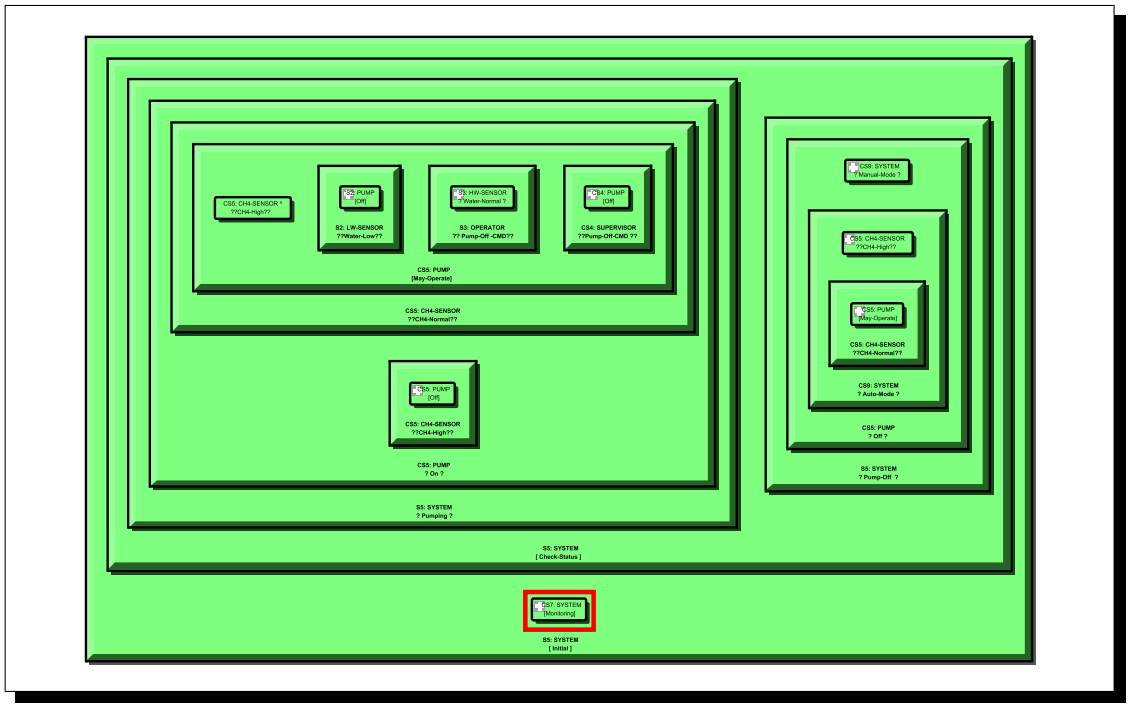


(a) Before

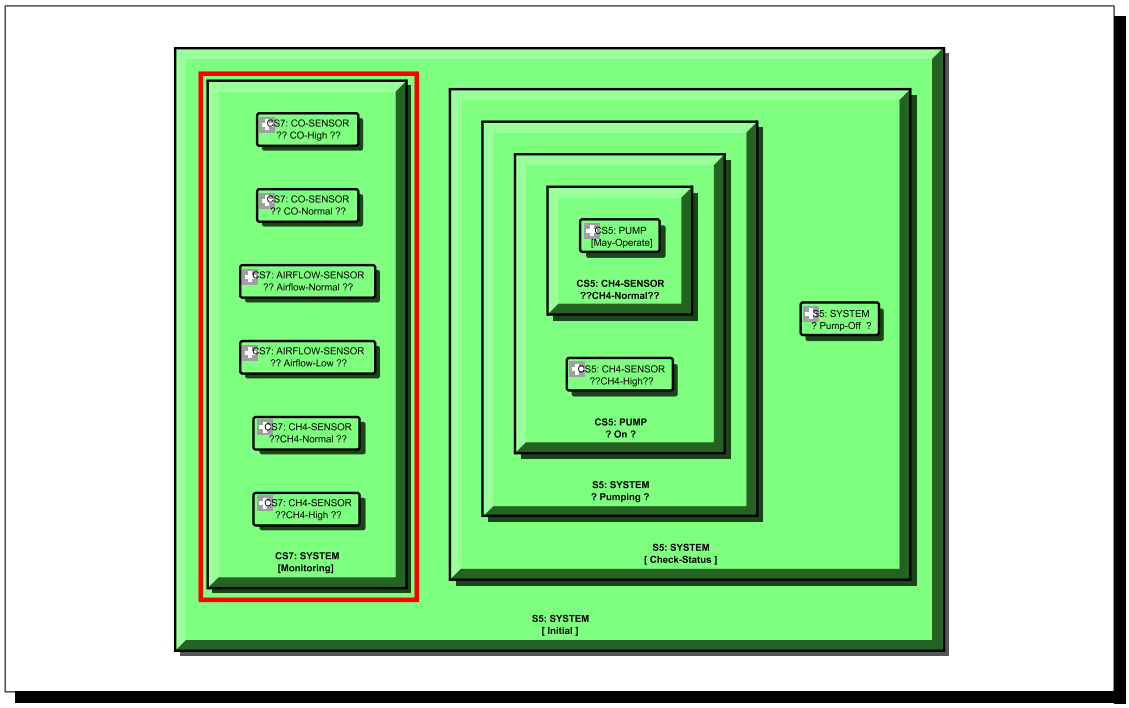


(b) After

Figure 4.5: Expanding the upper of the two left-most collapsed nodes in (a), indicated by a red box.



(a) Before



(b) After

Figure 4.6: Expanding the bottom-most collapsed node in (a), indicated by a red box.

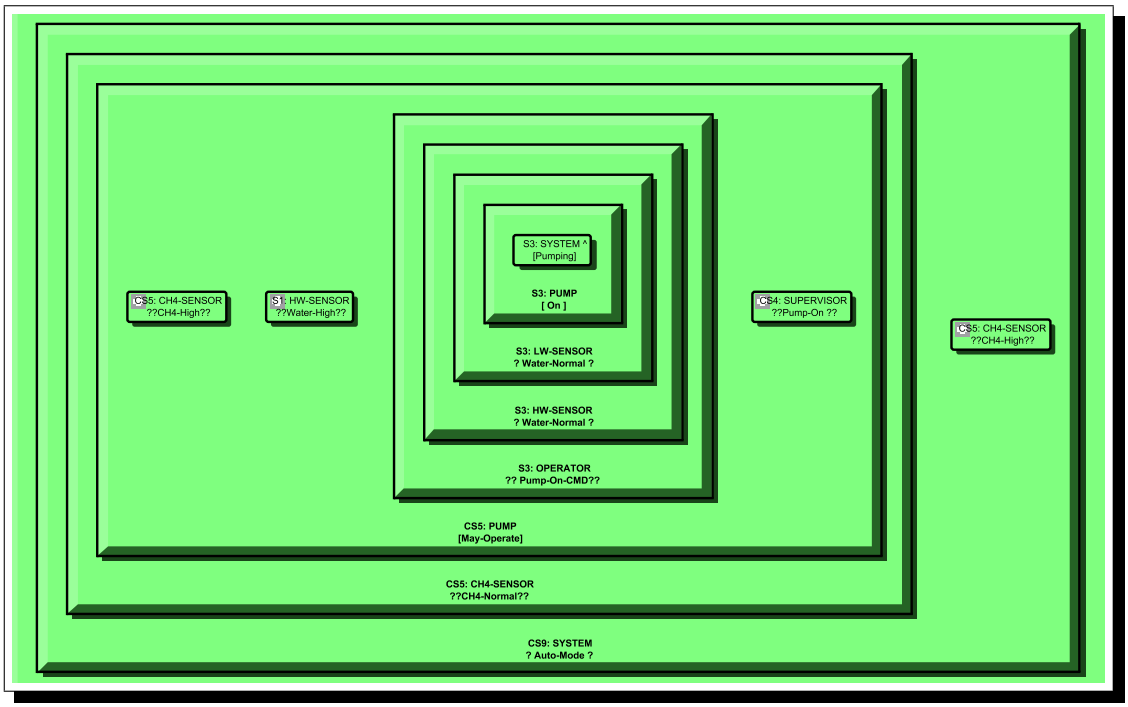
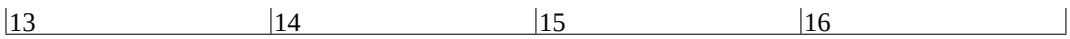


Figure 4.7: The effect of enabling level zooming for Figure 4.3.

place between them. This problem is compounded in an interactive system, as the "before" image is generally not available once the operation has occurred, and so it is not possible to directly compare the "before" and "after" visualisations. This section describes and demonstrates the animation techniques for h-v inclusion and tip-over tree layouts. By comparison to the static Figures 4.5, 4.6 and 4.7, it strongly reinforces the value of animation.

An animation method is required for each type of operation that can be performed, as well as for layout updates. In the inclusion tree style, expanding is performed by linearly interpolating the size of the collapsed node between its original collapsed size and its expanded size, followed by drawing the child nodes. Animated Figure 4.8 shows this animation. Collapsing is the time-reversed animation, that is, the expanded node's children are removed and the size of the node is then linearly interpolated to its original collapsed size. In both cases, changes in node sizes affects other nodes such that no occlusions occur; for example, as a node is expanded, its ancestor nodes are also expanded as necessary.

Expanding and collapsing nodes is rather more complicated in the tip-over tree style. The approach taken is to have the child nodes "slide" in or out of the parent node. For simplicity, only the expand operation is described, as the collapse operation is obtained simply by time-reversing



Animated Figure 4.8: Expanding a node using the inclusion tree style.

the expansion animation. In addition, only expansion to horizontal layout is considered, for the 'real center' h-v parent node strategy — the remaining variants are easy to obtain following the same principles. The path of each node in the expansion animation has two phases, such that it approximately follows the corresponding node-link edge in the tree. The first phase, slides the children down together out of the parent node. The second phase, separates the children, distributing them in x to their final position. The speed of each node is constant over its path. The size of each child node is linearly interpolated during the animation from the size of the parent node to the size of the child node. Animated Figure 4.9 shows an example of the tip-over node expansion animation.

In both the inclusion and tip-over styles, level zooming in and out is simply achieved by linearly interpolating the clipping rectangle of the display from the old pseudo-root node region to the new one.

Adjusting the layout requires changing the arrangement of one or more nodes from horizontal to vertical, or vice-versa. This is achieved by "rotating" the child nodes about the center of the node in question, as illustrated in Figure 4.10. Figure 4.11 illustrates the three different types of rotation our system uses:

1. *Linear*, which simply linearly interpolates the positions of the children from their

13 14 15 16

Animated Figure 4.9: Expanding a node using the tip-over tree style.

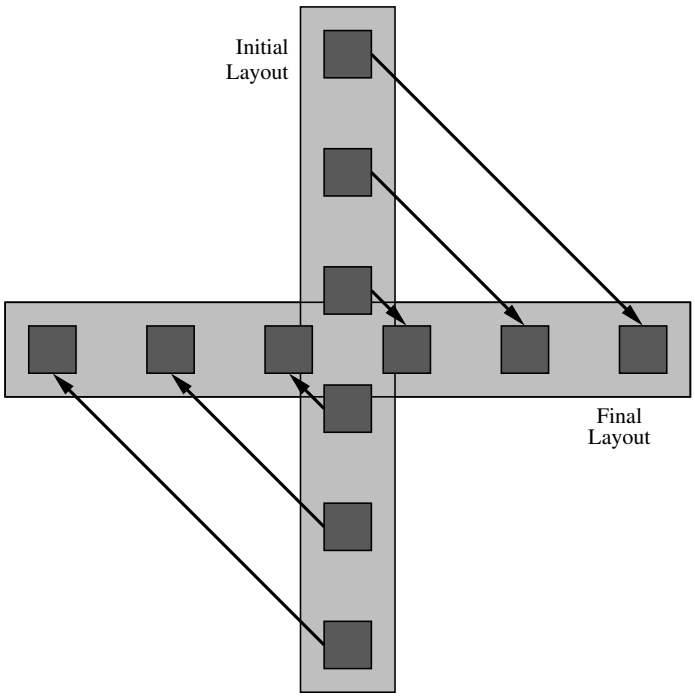


Figure 4.10: Rotating the child nodes to change a node arrangement from vertical to horizontal.

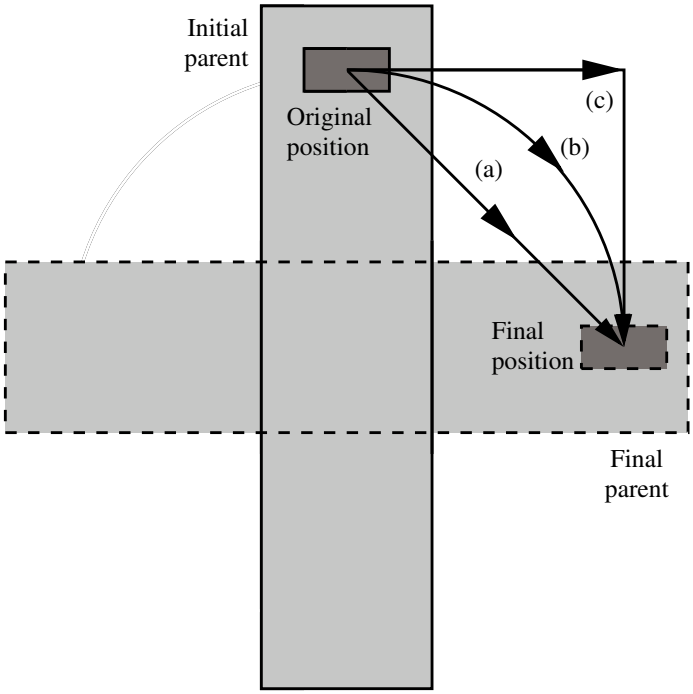
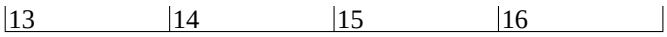


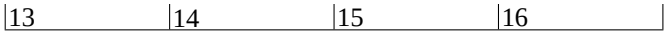
Figure 4.11: The different types of rotation available for updating the layout of a node, (a) linear, (b) circular and (c) orthogonal. The node is moved from its original position to its final position along one of the indicated paths.

13 14 15 16

Animated Figure 4.12: Linear rotation of a h-v node.



Animated Figure 4.13: Circular rotation of a h-v node.



Animated Figure 4.14: Orthogonal rotation of a h-v node.

initial location to their final location.

2. *Circular*, which interpolates the positions of the children along an elliptical arc.

3. *Orthogonal*, which interpolates the positions of the children along a "Manhattan path".

These rotation types are shown in Animated Figures 4.12, 4.13 and 4.14. Linear and circular rotations are subject to occlusions between moving siblings, although circular rotation is not as prone to it. Orthogonal rotation requires the nodes to traverse a longer distance, and can cause the size of the node to increase considerably during the animation, but has the advantage of completely avoiding occlusions.

Rotating the layout of a node between horizontal and vertical in the tip-over style, is similar to that in the inclusion style, except that the parent node and node-link edges must also be animated. This is achieved by applying the same rotation type to the parent node and each of the bends in the node-link edges, shown in Animated Figure 4.15.

Animated Figures 4.16 and 4.17 show the animated transitions of the static images in Figures 4.5 and 4.6 using orthogonal h-v node rotation. The difference in ease of understanding between the static images and animations is immediately apparent.

The animation file accompanying this thesis, `minepump-tour.wmv`, shows the animations used by the system on the Mine Pump DBT from Figure 2.17(b). It demonstrates a navigation through the DBT, illustrating the main features of the Structural Zooming system. The compressed version of the Mine Pump DBT from Figure 2.41 is not used, as it is not deep enough to exhibit level zooming. The data content measure used is the number of nodes present, with a maximum detail threshold of 12 nodes. This is unrealistically low, in order to illustrate the features of Structural Zooming. A side-effect of this is that some nodes are closed immedately upon the next node being expanded: this situation would be avoided with more realistic settings.

4.2 Arbitrary node layouts

The application of Structural Zooming to arbitrary inclusion layouts is now considered, in particular, those that use the Jewellery Box Inclusion Layout Algorithm (JBILA) described in Section 2.2.2.

One of the biggest problems with the Jewellery Box Inclusion Layout is that it is not *stable*. This means that a small change in the input tree may cause a large change in the resulting layout.

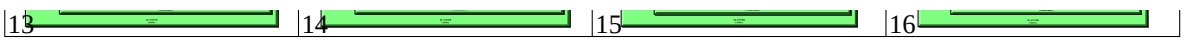
13 14 15 16

Animated Figure 4.15: Rotation of a h-v node in the tip-over style, using orthogonal animation.

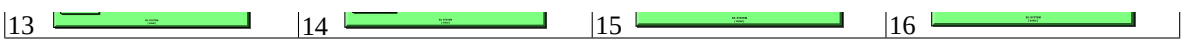
For example, Figure 4.18 shows a tree of six differently sized nodes laid out using the JBILA. The nodes are labelled A to F in decreasing order of size. Node D is now enlarged to be the largest node (as though it had been expanded). Figure 4.19(b) shows the result of using JBILA on this modified layout, which, as can be seen, bears little resemblance to the original layout in Figure 4.18.

Clearly, this situation is far from adequate for Structural Zooming. It is a requirement that the on-screen changes caused by a structural change are minimised. On its own, the JBILA makes no attempt to do this. Unlike the h-v inclusion layout style, there is no inherent structure which can be exploited for the layout transition animations.

Animated Figure 4.19(c) shows the transition from Figure 4.18 to Figure 4.19(b) animated using



Animated Figure 4.16: Animated transition of the operation shown in Figure 4.5.



Animated Figure 4.17: Animated transition of the operation shown in Figure 4.6.

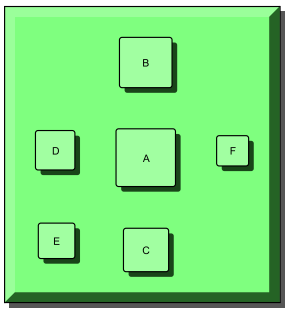
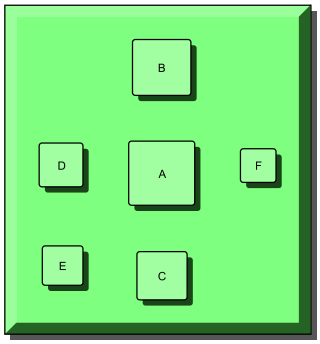


Figure 4.18: A JBILA layout of 6 differently sized nodes.

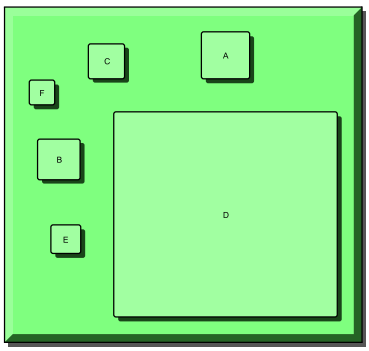
linear interpolation. This animation is clearly very hard to follow, and therefore is unsatisfactory for use in an interactive system. The poor suitability of linear interpolation for general graph animation has also been noted by Friedrich [54, 55].

We may use the results found in Friedrich's thesis [54], to construct a solution to the layout transition animation problem. Friedrich presents methods for *graph animation*, which demonstrate, given initial and final positions for a set of nodes in the 2D plane, how to obtain a pleasing animation of the transition. The methods investigated by Friedrich involve linear regression to find appropriate affine transformations, and clustering to find appropriate subsets of the graph to apply the affine transformations. The 2D affine transformations considered — translation, scaling, rotation and shearing — correspond to the 2D projections of manipulations of 3D objects in space — translation, scaling and rotation. The underlying assumption of this work is that these 2D affine transformations are easy for humans to follow and understand, since humans are familiar with the 3D manipulations to which they correspond. While this is yet to be experimentally verified with human subjects, Friedrich showed that the techniques work well with respect to some empirical measures of graph animation quality.

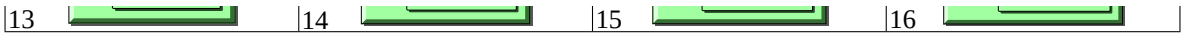
Although this method is useful for animating arbitrary and general changes in node positions, it is not appropriate to use it as the sole means for animating between layouts in Structural Zooming. The method deals with how to best mitigate the impact of the changes on the user's mental map: it assumes that the initial and final layouts are fixed and is not concerned with how they are obtained. By contrast, in Structural Zooming, the final layout is found by performing modifications and rerunning the layout algorithm on the initial layout. Friedrich's method makes no attempt to minimise the actual changes themselves, and simply animates the transition, even if a better final layout exists. In Structural Zooming, it is not particularly useful to have all the on-screen nodes



(a) Initial layout

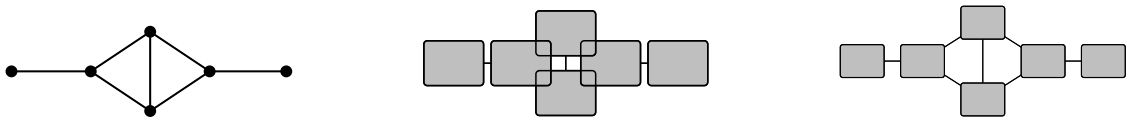


(b) Final layout



(c) Linearly animated transition

Figure 4.19: Node D from Figure 4.18 has been expanded, and the resulting tree laid out using JBILA.



(a) Point node layout

(b) Non-zero node sizes

(c) After applying FTA to (b)

Figure 4.20: The effect of using the Force Transfer Algorithm.

completely rearranging after every operation — no matter how pleasing the transition animation. As such, Friedrich's graph animation methods may be useful for enhancing the animations in a Structural Zooming system, but not on their own — more is required.

This prompts the consideration of a method which specifically aims to minimise the changes induced by an operation. The most straightforward way to do this is to limit the number of nodes which move, and the amount by which they move. One such way, is to apply the *Force Transfer Algorithm* (*FTA*) [77] (or its predecessor, the *Force Scan Algorithm* (*FSA*) [42, 89]) to the display, after performing each Structural Zooming operation, rather than performing a full layout. The FTA and FSA are techniques for resolving node–node and node–edge occlusions in graph drawing. Typically, graph drawing algorithms treat nodes as points in the plane, as shown in Figure 4.20(a). However, in most real-world applications, nodes have some associated information, such as text or images. To display this information in the graph drawing, requires nodes to be drawn in a bounded finite region of the plane — usually as rectilinearly aligned rectangles. As the graph drawing algorithm has neglected the non-zero area of the nodes, drawing rectangles centered at the locations of the (point) nodes can cause these nodes to occlude edges and each other, as shown in Figure 4.20(b). The FTA and FSA methods are incremental graph layout postprocessing techniques for resolving this problem. They use a force-based approach, similar to that used in force-directed graph layout [32, 40, 56], to determine how to move each node as little as possible whilst resolving any occlusions and preserving the layout as much as possible. This is achieved by preserving the *orthogonal ordering* of the layout, which considers only the order of the nodes in the x and y directions. At a simple level, the algorithm works as though "springs" are placed between all pairs of adjacent nodes, forcing overlapping nodes to separate and preventing new occlusions from occuring. The resulting layout closely resembles the original, without the occlusions, as shown in Figure 4.20(c).

The FTA approach can be adapted for use with Structural Zooming. Rather than following each Structural Zooming operation with a full layout of the entire visible graph, the previous layout is maintained and the operation is appled. This causes the sizes of some nodes to change, and then the Force-Transfer Algorithm can be applied to resolve any resulting occlusions. This gives a purely incremental approach to the problem, where the resulting layout closely resembles the previous layout. Consequently, it is easy to animate the movement of the nodes using simple linear interpolation, or even using the path effected on the nodes by the forces present in the FTA.

Whilst such an incremental strategy is pleasing from the point of view of a mental map and transition animation, it sacrifices the quality of the layout. The only time a layout is performed is initially, after this, the layout is merely modified with each successive Structural Zooming operation. These modifications pay no regard to the layout quality or to any specific graph drawing aesthetic criteria, but only consider the orthogonal order of the nodes. This means that the quality of the layout can be expected to decrease after each operation and these decreases accumulate over many Structural Zooming operations, thus causing the quality of the layout to deteriorate. Since good layout quality is an explicit goal of Structural Zooming, this is clearly not acceptable. This problem may be mediated by monitoring the quality of the layout with some measure, and then performing a full layout when the quality decreases below a predefined threshold of acceptability. This lower limit may be relative to the quality of the last full layout algorithm results, or perhaps relative to the quality of the optimal layout for the current display. However, this full layout would then require a full animation solution, such as using Friedrich's previously discussed graph animation techniques. This means that the incremental animation approach merely defers the problems which arise from changing from one full layout to another, rather than solving them.

4.2.1 Stable jewellery box inclusion layout algorithm

The traditional approach to the stability of graph layouts is the use of *layout constraints* [23]. These are additional restrictions on the positions of nodes, based on their previous positions. However, these additional constraints are generally represented as linear equations, which means that they are best integrated with graph drawing algorithms that compute layouts with techniques such as linear programming and integer linear programming. In such cases, the additional layout stability constraints are simply expressed as part of the actual graph drawing problem instance, and solved without any additional consideration from the graph drawing algorithm.

This thesis presents a similar approach to layout stability. The JBILA is fully run after each

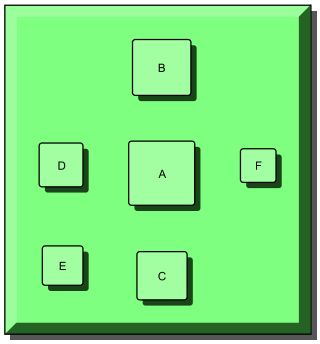
Structural Zooming operation, however, it has been extended to enhance its stability. This modified JBILA is the *Stable Jewellery Box Inclusion Layout Algorithm* (*SJBILA*). Since it is based on the JBILA, a good quality layout is assured, and because the layout only changes by a small amount after each operation, simple animation techniques such as linear interpolation may be used.

A key observation is that the JBILA does not return a unique optimal layout, rather, it is an heuristic that returns one of many possible solutions. The solution found depends on the decisions made during the course of the algorithm. The goal is to guide the JBILA into selecting a layout that is as similar as possible to the previous layout. The first time the SJBILA is run there is no previous layout, and so the layout proceeds identically to the standard JBILA.

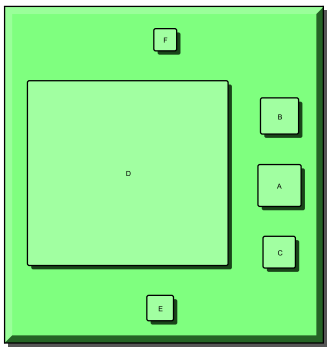
First, the order in which the nodes are placed does not change. This prevents nodes that are expanded (and thus larger than previously shown) from "jumping" to the start of the list of nodes. For example, consider expanding the i-th largest node so that it becomes the largest node. (Here the nodes are labelled according to their placement order in the initial layout, $1...n$.) JBILA places this node i first, and following that places the nodes $1...i-1, i+1...n$ around node i, potentially creating a very different layout. By preserving the placement order, it is ensured that nodes 1 to $i-1$ have the same layout as previously, giving the same "base" layout for the placement of node i and nodes $i+1...n$.

Figure 4.21(b) shows the tree from Figure 4.18 after applying JBILA with the same node placement order as in Figure 4.18. Note that in Figure 4.21(b) the relative placement of nodes A to C is the same as in Figure 4.18, but their placement in Figure 4.19(b) is substantially different. Animated Figure 4.21(c) shows the linearly animated transition from Figure 4.18 to Figure 4.21(b). We can observe that there is still some occlusion, but considerably less than in Animated Figure 4.19(c).

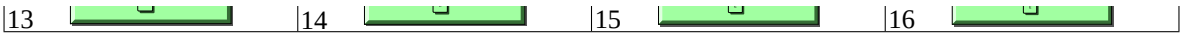
Secondly, the selection of possible node placements is restricted to those which preserve the orthogonal ordering of the node with respect to its immediate geometric neighbours. This orthogonal ordering is defined as follows. The orthogonal ordering of node v with respect to (or "relative to") node u is given by which of the eight rectilinear regions around u the centre of node v lies in. This is illustrated in Figure 4.23, showing the eight possible regions and a node v which lies in the NE region, based on its centre position. Given a new location of node v, say, v', the orthogonal ordering of v (relative to u) is preserved if and only if v' lies in the same region as v or an adjacent region. Regions are adjacent if they share a common bounding edge. For example, region E is adjacent to NE and SE, but not to S, SW, W, NW or N. This means that the node v shown in Figure 4.23 may to move anywhere in regions N, NE and E while still preserving the orthogonal ordering.



(a) Initial layout

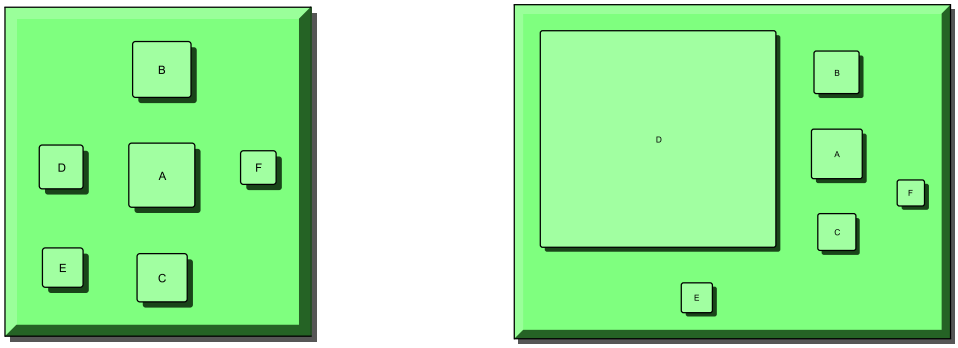


(b) Final layout



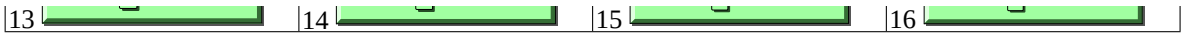
(c) Linearly animated transition

Figure 4.21: As for Figure 4.19, except that the nodes are placed in the same order as in Figure 4.18.



(a) Initial layout

(b) Final layout



(c) Linearly animated transition

Figure 4.22: As for Figure 4.19, except that the stable JBILA has been used.

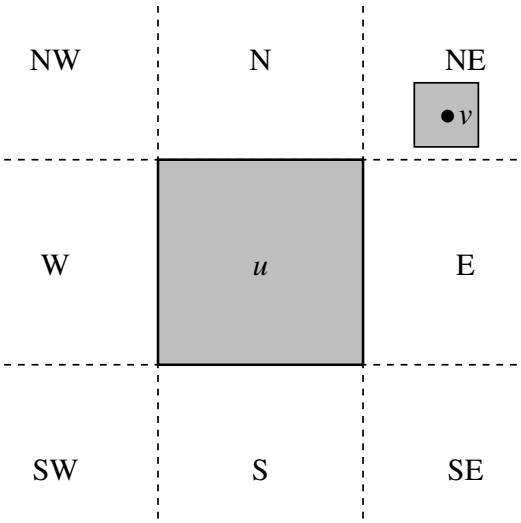


Figure 4.23: The eight orthogonal ordering regions surrounding node u, labelled according to the points of the compass, with node v in the NE region.

During the algorithm, candidate points which do not preserve the orthogonal ordering for each of their neighbours, are not considered for placement. To find the "neighbour" nodes, the algorithm examines the Delaunay triangulation maintained by the JBILA. It finds the triangles that are within a distance of k from the triangle to which the candidate point belongs. The neighbours are the nodes at the corners of these triangles. The parameter k controls how strictly the orthogonal ordering is checked, larger values indicate that the candidate point is compared against more neighbour nodes. The value of k should not be too large, as being too strict in the preservation of the orthogonal ordering will cause many candidate points to be completely rejected. It is desirable to check the nearest neighbours, but since the goal is the adjustment of the layout, some leniency must be granted. This system uses $k = 0$, that is, only the three immediate neighbours are used. Applications requiring stronger orthogonal ordering preservation are free to use $k = 1$, $k = 2$ or higher, as necessary.

Figure 4.22(b) shows the tree from Figure 4.18 laid out using the JBILA with preserved node layout order and respecting orthogonal ordering. Note that, in this case, the nodes E and F have been placed in locations closer to their original situation. This is despite their "base" layout of nodes A–D having changed. The result is a layout which much more closely resembles the previous display. Animated Figure 4.22(c) shows the linearly animated transition from Figure 4.18 to Figure 4.22(b). In this case, there are no longer any occlusions at all, and the animation is similar to that which would be obtained using the FTA or FSA algorithms. The transition is clearly superior to that shown in Figure 4.19, even though both make use of linear interpolation.

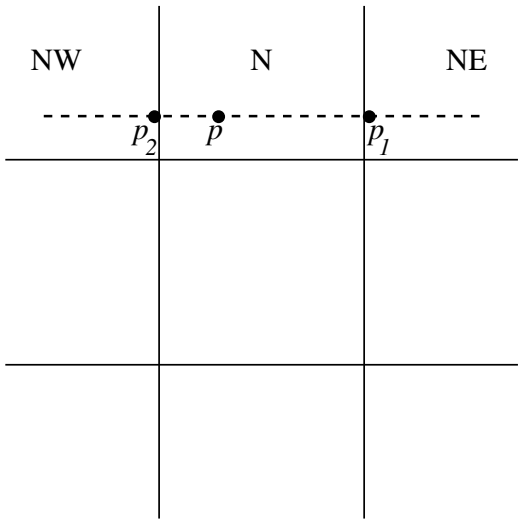


Figure 4.24: The additional candidate positions p_1 and p_2, from an original position p in the N region.

However, due to the way in which JBILA selects candidate positions for node placement, it is possible for a node to have no candidate positions which preserve the orthogonal ordering. In this case, we can re-examine the candidate positions (in order), and use a heuristic to generate additional candidate positions for each. The additional positions are checked for orthogonal ordering preservation, and positions which fail are skipped. Of the remaining positions, the one which is the shortest distance from the previous node position is selected. In the unlikely event, that all the additional candidate positions are skipped, the algorithm simply selects the candidate position which is the shortest distance from the previous node position.

The additional candidate positions are found as follows. If the actual position p is in region N, then we choose new positions p_1 and p_2 by holding the y coordinate of p constant and "sliding" it across until it just enters the NW and NE regions, as shown in Figure 4.24. Similarly for points in regions S, W and E. For points in region NE, the two points p_1 and p_2 are found by "sliding" towards regions N and E, as shown in Figure 4.25. Similarly for regions NW, SE and SW.

Thus, a stable layout is very important when considering animated changes to the graph being laid out. Often, there are places in graph layout algorithms where equivalent choices could be made; in such cases it is best to consider the stability of the layout, and aim to make the layout change as little as possible.

It is noted that this technique does not prevent node occlusions during the animation. However, due to the similarity of the layouts, and the types of operations involved, such occlusions are usually

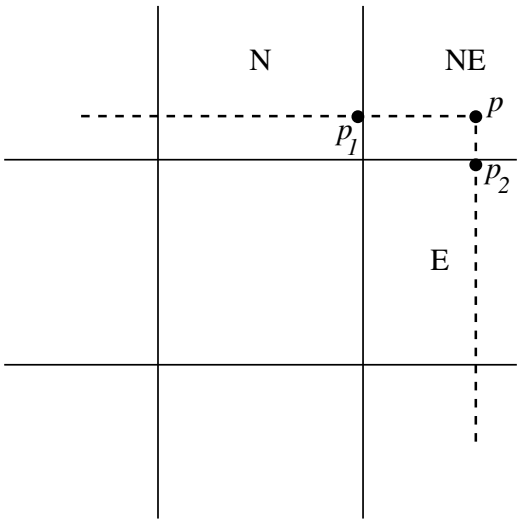


Figure 4.25: The additional candidate positions p_1 and p_2, from an original position p in the NE region.

only small (compared to the size of either node), and brief (compared to the length of the overall animation). It is also noted that, in rare circumstances, the technique does not prevent the entire layout from rearranging. In such cases, the user's mental map is usually lost. However, the situtation is no worse than with the standard JBILA algorithm. Techniques such as Friedrich's graph animation [54] can be used to help improve the situation (compared to linear interpolation). In such cases, it may also be beneficial to break the animation of the node positions into several sequential operations. Although it would make the overall animation longer, users would be more forgiving, as it would happen relatively infrequently. Deciding how to partition the nodes into groups is a separate issue which would need to be addressed. Psychological research shows that people can track up to four differently moving objects at any given time [126], and that objects with similar motions are perceived as being grouped together [17, 34, 97, 99, 105]. This means that the nodes should be grouped so that there are at most four distinct velocity vectors in the animation (within some tolerance).

4.3 Orthogonal edge animation

As described in Chapter 2, a clustered graph is an inclusion tree with the addition of edges between nodes. The application of Structural Zooming to clustered graphs utilises the previous application of Structural Zooming to inclusion trees, with the node-link edges of the clustered graph treated as a separate additional consideration. Two aspects must be considered: the edge routing algorithm,

used to determine the layout of the edges in the presence of the already placed inclusion tree nodes, and the edge animation strategy, used to animate changes in the edges due to Structural Zooming operations.

Recall from Section 2.1.2 that an orthogonal edge layout is simply an ordered list of real numbers, which are the alternating x and y ordinates of its segments (and endpoints). An *orthogonal edge animation* is defined as a function

$$f : [0, 1] \rightarrow \mathcal{E},$$

where $\mathcal{E}$ is the set of all possible orthogonal edge layouts, given by

$$\mathcal{E} = \bigcup_{k \in \mathcal{N}} \mathbb{R}^{k+2},$$

where $\mathcal{N}$ is the set of natural numbers. The parameter to f, denoted by t, represents time in the animation; $t = 0$ indicates the start of the animation and $t = 1$ indicates the end. The orthogonal edge layout $f(0)$ is called the *initial frame* or *initial layout*, and $f(1)$ is called *final frame* or *final layout* of the animation. The remaining orthogonal edge layouts $f(t), 0 < t < 1$ are called the *intermediate frames* or *intermediate layouts*. The *ideal animation* is defined over all time points in the range 0–1. However, as computer animation is achieved using apparent motion perceived between discrete images (frames), the *actual animation*, or simply the *animation*, is a finite collection of orthogonal edge layouts, obtained by sampling the ideal animation at regular intervals in time. Thus, an orthogonal edge layout is animated by changing the values of its segments over time. An animation is said to be *smooth* if f is in C_1, that is to say, the trajectories of each of the edge segments are first-order differentiable on the closed interval $t \in [0, 1]$. It is worth noting that this definition allows for the possibility of the addition and removal of segments, since the number of segments k may be different in each intermediate frame. In this case, an animation is said to be smooth if the length of a removed edge segment tends to 0, and similarly for inserted edge segments. Another way of considering this is to use "degenerate" segments to ensure that the number of segments does not change throughout the animation. Section 4.3.3 describes this situation in much more detail, and it is sometimes assumed (for simplicity, and without loss of generality) in the following sections that all the frames have the same number of segments.

Although f defines an orthogonal edge animation, it is difficult to compare the *quality* of two or-

thogonal edge animations f_1 and f_2. This is because there is no clear indication of what quantitative properties the "optimal" or "best" animation should have.

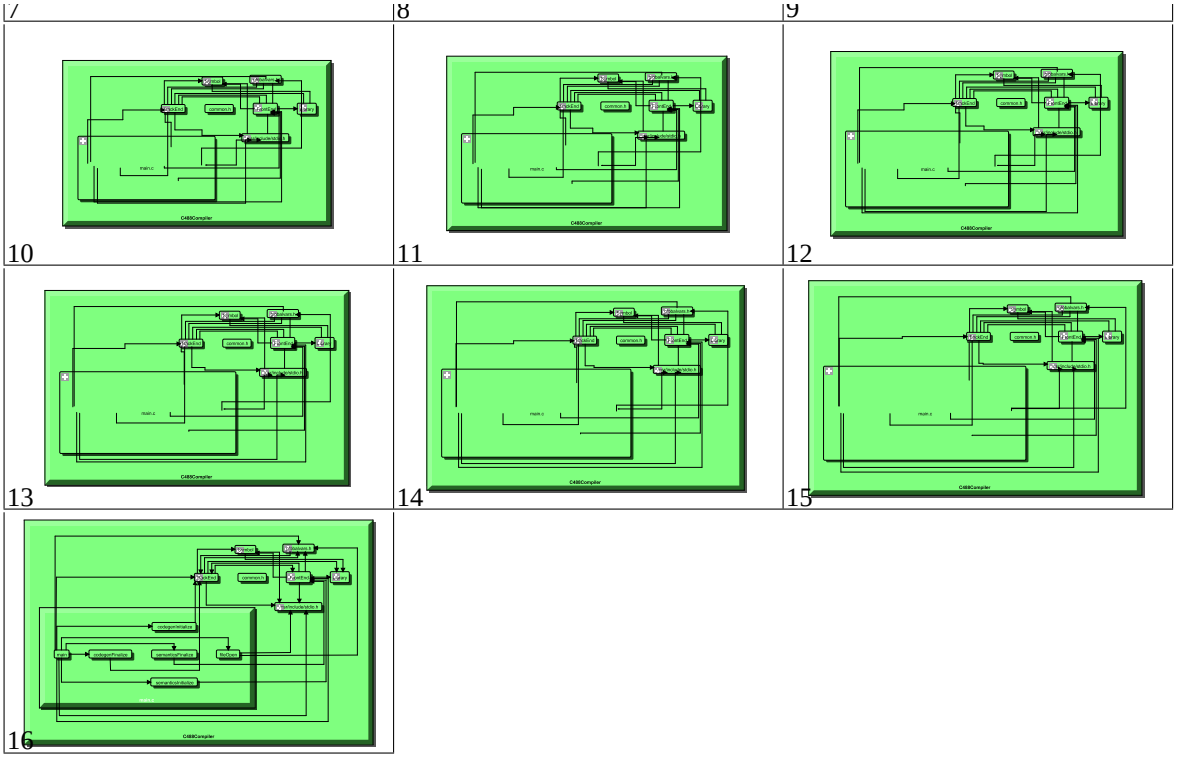
Different edge animations can differ in quality. For example, Figures 4.29 and 4.31 show two alternate animations between the same initial and final edge layouts — clearly Figure 4.31 is simpler and easier to understand than Figure 4.29. There are an infinite number of possible animations between any pair of initial and final orthogonal edge layouts. The situation is similar to the question of determining the quality of a graph drawing. The field of graph drawing has devised various *aesthetics*, which are quantitative measures of particular qualitative aspects of a graph drawing [125]. The hypothesis is that optimising the aesthestics yields better graph drawings. Exactly which aesthetics are optimised is generally a domain-specific issue, and thus is a choice made by the graph drawing algorithm designer. We approach the problem of determining an orthogonal edge animation in a similar way. The function $\mathcal{M} : \mathcal{E} \to \mathbb{R}$ is a *quality measure* (i.e. aesthetic), which takes an orthogonal edge layout and returns a numeric quality value. Several orthogonal edge layout quality measures are described in Section 4.4.

The *orthogonal edge animation problem* is defined as follows:

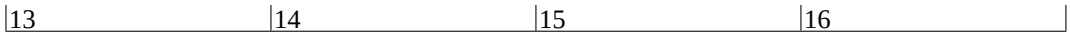
> **Orthogonal Edge Animation Problem** (OEAP): Given an initial orthogonal edge layout $E_0 \in \mathcal{E}$ and final orthogonal edge layout $E_1 \in \mathcal{E}$, find an orthogonal edge animation f such that $f(0) = E_0$, $f(1) = E_1$ and the value
>
> $$\int_{t=0}^{1} \mathcal{M}(f(t))$$
>
> is minimised over all possible choices of f.

Animated Figure 4.26 shows an example of the animation that results from expanding a node in a clustered graph, that of the `c488` compiler (described in more detail in Chapter 6). The node placement is determined by the SJBILA, and the orthogonal edge animation is determined using the methods presented in this section.

Throughout this section, edge layouts are assumed to be *normalised*, which means that they always begin with an x segment, and end with a y segment. A justification of this and method for ensuring it is presented in Section 4.3.3.



Animated Figure 4.26: An example animation of expanding a node in the `c488` compiler, described in Chapter 6.



Animated Figure 4.27: A simple linear edge animation.

4.3.1 Linear edge interpolation

A *linear edge interpolation* or *linear edge animation* is an orthogonal edge animation where the value of each segment is moved from its initial value to its final value, using linear interpolation. All the segments are moved concurrently, and the lengths of the segments are automatically adjusted in order to maintain connectivity. For initial and final layouts given by

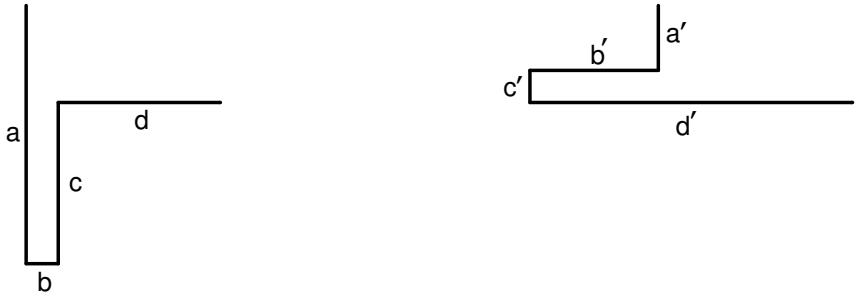
$$\begin{aligned} E_0 &= (x_1, y_1, x_2, y_2, ..., x_n, y_n) \\ \text{and} \quad E_1 &= (x'_1, y'_1, x'_2, y'_2, ..., x'_n, y'_n), \end{aligned}$$

the segments of an intermediate orthogonal edge layout at time t are given by the expression

$$x_i(t) = x_i + t(x'_i - x_i)$$

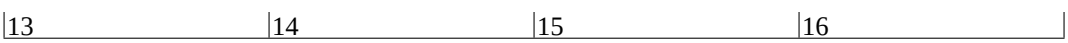
for $1 \leq i \leq n$, and similarly for y. Here it is assumed that all the orthogonal edge layouts have the same number of segments. Differing numbers of segments are described below in Section 4.3.3. Animated Figure 4.27 shows a simple example of a linear edge animation.

Now consider the orthogonal edge animation problem instance, in which an orthogonal edge



(a) Initial edge layout (b) Final edge layout

Figure 4.28: An orthogonal edge animation problem instance.



Animated Figure 4.29: Linear animation solution to the orthogonal edge animation problem instance shown in Figure 4.28.

(a) (b)

Figure 4.30: Frames in Figure 4.29 which contain segments of zero length.

13 14 15 16

Animated Figure 4.31: Improved animation solution to the orthogonal edge animation problem instance shown in Figure 4.28.

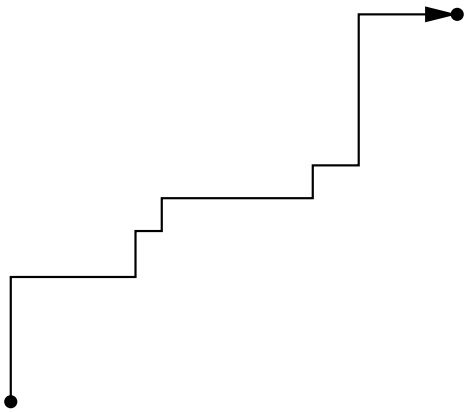


Figure 4.32: A NE-monotone orthogonal edge layout.

layout must be animated between initial and final layouts as shown in Figure 4.28. If the position of each orthogonal segment is animated using linear interpolation, the result is shown in Animated Figure 4.29. The relative positions of b, b', c, c' mean that during the animation the edge becomes self-intersecting between segments a and d. Further, there are two instantaneous points in the ideal animation, when the length of segments b or c is zero, shown in Figure 4.30. Intuitively, it seems that any ambiguities (as defined in Section 2.1.2) are highly undesirable during edge animation, and detrimental to the users' comprehension of the changes being animated. For example, Animated Figure 4.31 shows an animation of the instance from Figure 4.28, which avoids these problems. This animation appears to be easier to follow, although it does have additional disadvantages, notably that it introduces additional edge segments (and therefore edge bends) during the course of the animation.

4.3.2 Monotone edge layouts

An orthogonal edge layout is *monotone* if the sequence of x coordinates of the bends increases or decreases monotonically, and similarly for the y coordinates. In the same way that edges are directed[1], edge segments are also directed from a source bend (or node) to a target bend (or node), such that the chain of edge segments may be traversed from the source node to the target node. For orthogonal edges, this gives four possible edge segment directions, corresponding to increasing/decreasing horizontal/vertical segments. These *segment orientations* are labelled according to

[1]Note that undirected edges can be considered to be directed by consistently choosing one node to be the source and one node to be the target. This may be done, for example, by labelling the nodes from 1 to n, and then directing edges from the node with smaller label to that with larger label.

	NE	NW	SW	SE
NE	similar	partial	dissimilar	partial
NW	partial	similar	partial	dissimilar
SW	dissimilar	partial	similar	partial
SE	partial	dissimilar	partial	similar

Table 4.1: Monotonic similarity between orthogonal edge layouts.

the points of the compass, that is, N (North) for increasing vertical, S (South) for decreasing vertical, E (East) for increasing horizontal and W (West) for decreasing horizontal. We can use this to represent the "topology" or *edge orientation* of an edge as a string of segment orientations. For example, assuming that the edge is directed in order of segments a, b, c, d, the edge orientation of Figure 4.28(a) is SENE, and the edge orientation of Figure 4.28(b) is SWSE. Clearly, an edge layout is monotone if and only if the vertical segments of its orientation are either all N or all S (or neither), and the horizontal segments are either all E or all W (or neither). For example, Figure 4.32 shows a monotone edge layout with orientation NENENENENE, or $(\mathrm{NE})^5$, and we say that this edge is *NE-monotone* or that its *monotonicity* is NE. An orthogonal edge layout may be monotone in only the x or y direction, in which case it is said to be *N-*, *S-*, *E-* or *W-monotone*, as appropriate. Two monotone orthogonal edge layouts are *similar* if they have the same type of monotonicity, *partially similar* if their monotonicities differ in one axis, and *dissimilar* if their monotonicities differ in both axes (ie. their monotonicities are different). This is summarised in Table 4.1.

A key observation is that linear interpolation of edge segments is guaranteed to avoid the problems shown in Figure 4.29 (that is, edge self-intersection and ambiguity), if the initial and final edge layouts are monotone and similar. This is shown in the following theorem.

Theorem 4.3.1 *In a linear interpolation between two XY-monotone orthogonal edge layouts, intermediate edge layouts are also XY-monotone.*

Consider (without loss of generality (WLOG)) two NE edge layouts with the same number of segments,

$$(x_1, y_1, x_2, y_2, ..., x_n, y_n)$$
$$\text{and} \quad (x'_1, y'_1, x'_2, y'_2, ..., x'_n, y'_n).$$

(Differing numbers of segments can be handled with degenerate segments,

as described below in Section 4.3.3.) The monotonicity means that $x_1 \geq x_2 \geq x_3 \geq ... \geq x_n$ and $x'_1 \geq x'_2 \geq ... \geq x'_n$, and respectively for y and y'. That is, the segments are ordered in x and y, with the same order in the initial and final layouts. The edge layout is given by linear interpolation, thus the i-th x coordinate at time $0 \leq t \leq 1$ is

$$x_i + t(x'_i - x_i) = (t-1)x_i + tx'_i,$$

and similarly for y. Consider the i-th and j-th segments at time t, where $i > j$. Since $x_i \geq x_j$ and $x'_i \geq x'_j$, it is easy to see that any linear combination of x_i and x'_i must be greater than or equal to the same linear combination of x_j and x'_j, that is, $ax_i + bx'_i \geq ax_j + bx'_j$. By choosing $a = t - 1$ and $b = t$, we obtain

$$(t-1)x_i + tx'_i \geq (t-1)x_j + tx'_j$$

for any i, j, t; and respectively for y. Therefore, in a linear edge animation between two edge layouts with the same monotonicity, all intermediate edge layouts are also monotone in the same way.

The situation for linearly animating between partially similar orthogonal edge layouts is analogous, except that it proceeds in only one of x or y, rather than both.

Theorem 4.3.2 *In a linear interpolation between two partially similar monotone orthogonal edge layouts, the intermediate edge layouts are x or y monotone, according to the monotonicity shared by the initial and final layout. That is, between XY- and XZ-monotone orthogonal edge layouts, intermediate edge layouts are X-monotone, and between XY- and ZY-monotone orthogonal edge layouts, intermediate edge layouts are Y-monotone.*

Consider WLOG the initial and final edge layouts to be NE- and SE-monotone. Using the same argument as Theorem 4.3.1, but only for the x values, gives the following condition for intermediate edge layouts:

$$(t-1)x_i + tx'_i \geq (t-1)x_j + tx'_j$$

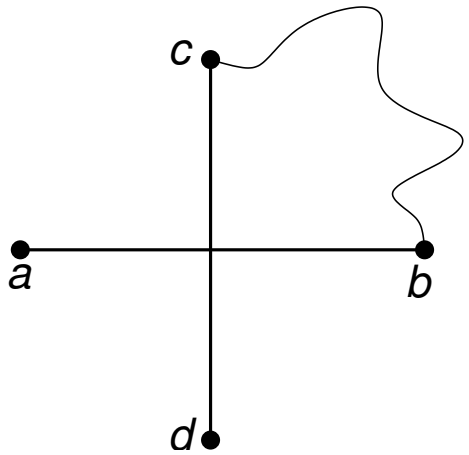


Figure 4.33: The general form of a self-intersection.

for any i, j, t. Thus all the intermediate edge layouts are E-monotone in x.

Furthermore, the following lemmas can also be proved.

Lemma 4.3.3 *Monotone orthogonal edge layouts do not self-intersect.*

The contrapositive of this is that edge layouts that self-intersect are non-monotone. The general form of edge layout self-intersection is shown in Figure 4.33, where the horizontal segment ab intersects the vertical segment cd, as it is not possible for two horizontal (or two vertical) segments to intersect. A "glancing" intersection is where one of the endpoints a, b, c or d lie on the opposite segment, for example (WLOG), where a lies on cd. This is essentially the same situation, as the endpoint a may simply be extended beyond the intersecting segment cd by some arbitrarily small δ. To be an edge, it must be that one of ac, ad, bc or bd are joined by some path of segments. As shown, we assume WLOG that b, c are joined and the edge is directed from a to d via b and c. Now ab has orientation E. Since the x value of cd is less than that of b, it must be the case that at least one W segment exists in the path bc. Thus, the edge layout is non-monotone in x. A similar argument holds in the y direction.

Lemma 4.3.4 *Monotone orthogonal edge layouts do not self-occlude.*

Consider WLOG a NE-monotone orthogonal edge layout. We now examine

the i-th vertical segment; the same argument holds in the y direction for the horizontal segments. The monotonicity property means that $x_i \leq x_{i+k}$ for positive integers $k < n - i$.

In the case where $x_i < x_{i+1}$, clearly none of the segments after i can have the value x_i. Since a necessary condition for self-occlusion is that the two segments have the same value, self-occlusion does not occur in this case.

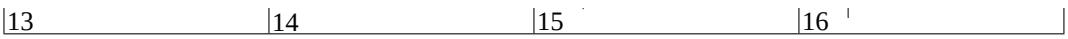
In the other case, $x_i = x_{i+1} = x_{i+2} = \ldots = x_{i+j} < x_{i+j+1}$, for $j < n - i - 1$. Here, the segments for $1 \leq k \leq j$ have the value x_i but because they are consecutive, they are merely degenerate, not self-occluding. The remaining segments (that is, for $j + 1 \leq k \leq n - i$) cannot have the value x_i, since $x_i < x_{i+j+1}$ as in the previous case. In fact, this shows that the previous case is simply the special case of $j = 0$.

Since i has been chosen arbitrarily, the edge must be free of self-occlusions.

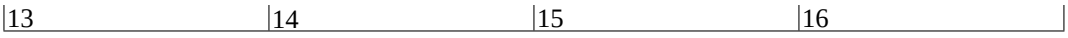
When Theorems 4.3.1 and 4.3.2 are combined with Lemmas 4.3.3 and 4.3.4, it is clear that any linear animation between similar or partially similar monotone orthogonal edge layouts do not contain ambiguities.

Animated Figure 4.34 shows an example linear animation from a SE- to NE-monotone orthogonal edge layout.

More care must be taken when animating dissimilar monotone orthogonal edge layouts. Animated Figure 4.35 shows a linear animation between two dissimilar monotone edges (SE and NW), where self-intersection occurs during the animation. This is a counter-example to the assertion that dissimilar monotone edges may be linearly animated without ambiguities, as is the case for similar and partially similar monotone edges. This problem is solved by defining a simple pseudo-linear edge animation strategy which reduces the dissimilar case to two partially similar cases. This is achieved by finding an intermediate edge layout C, such that C is partially similar to both the initial layout A and final layout B. For example, if A is NE and B is SW, then C is either NW or SE. It is now possible to animate from A to C, and then from C to B, each using linear interpolation. All that remains is to find a suitable layout C. This is easily achieved by combining the x segments from layout A with the y segments from layout B (or, equivalently, the y segments from layout A with the x segments from layout B). This is illustrated in Figure 4.36. Animated Figure 4.37 shows



Animated Figure 4.34: A linear animation from a SE-monotone to a NE-monotone edge layout.



Animated Figure 4.35: A linear animation from a SE-monotone to a NW-monotone edge layout, showing self-intersection in frames 9 and 10. The initial layout is $(x{=}0, y{=}0, 0, -8, 4, -16, 20, -16)$ and the final layout is $(x{=}0, y{=}0, 0, 2, -7, 10, -8, 10)$.

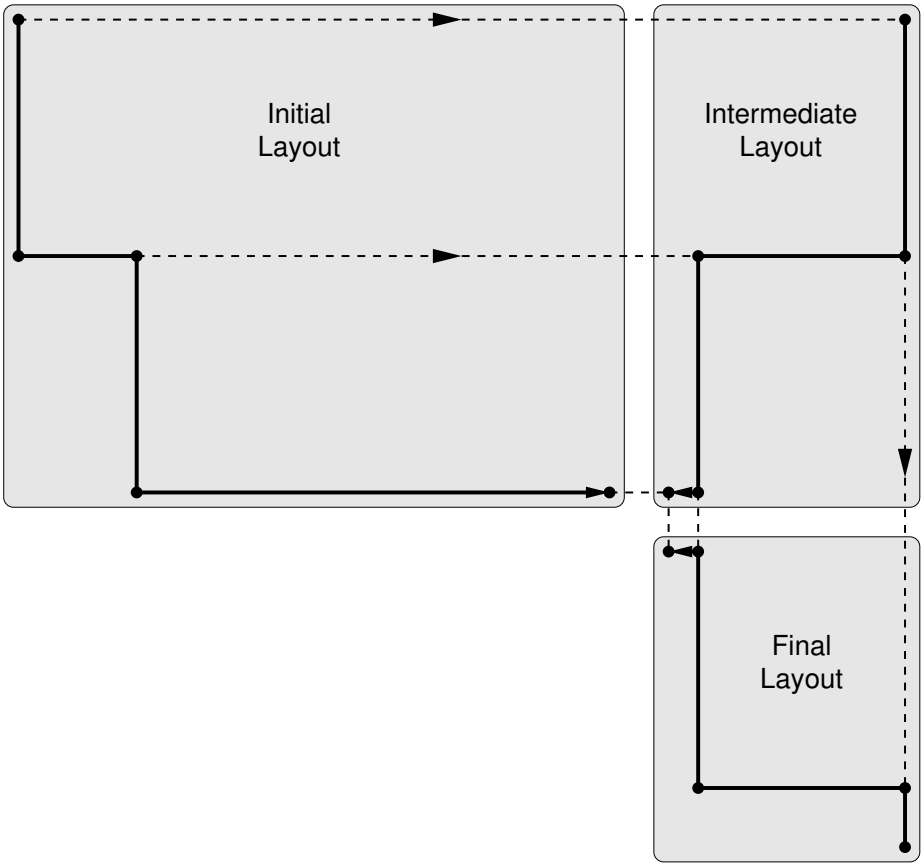
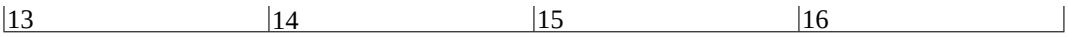


Figure 4.36: The SE-monotone to NW-monotone animation from Animated Figure 4.35, animated using an intermediate SW-monotone layout. The intermediate layout is $(x{=}0, y{=}0, 0, -8, -7, -16, -8, -16)$.

this technique applied to the example from Animated Figure 4.35.

4.3.3 Adding and removing segments

It is possible to use linear interpolation for edge layouts where the initial and final layouts have differing numbers of segments. This can be achieved by inserting *degenerate* edge segments at appropriate positions in the edge layout with fewer segments. These degenerate segments have zero length, and must be added in adjacent pairs in order to preserve the alternating horizontal-vertical structure of the orthogonal edge layout. For example, consider the edge $(x=0, y=0, 100, 100)$. A degenerate segment can be inserted halfway along the first segment, giving the edge $(x{=}0, y{=}0, 50, 0, 100, 100)$. This edge has an additional bend at $(0, 50)$, however, it appears visually identical to the original edge, as the vertical segment at $x{=}50$ has horizontal segments at $y{=}0$ on both sides. It is also possible to insert two degenerate segments at the bend in the original edge, giving $(x{=}0, y{=}$



Animated Figure 4.37: The orthogonal edge animation problem instance from Figure 4.35, animated using an intermediate partially similar orthogonal edge layout.

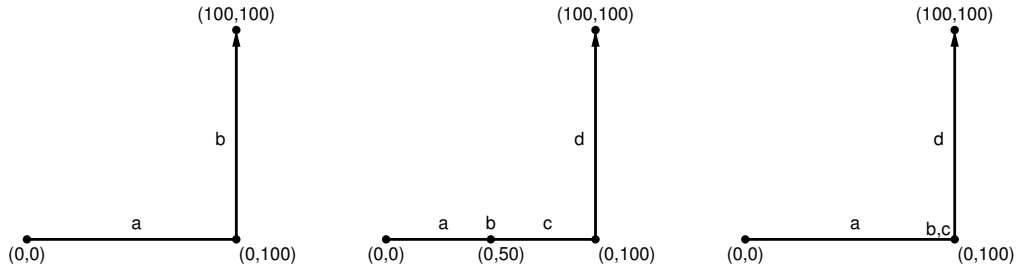
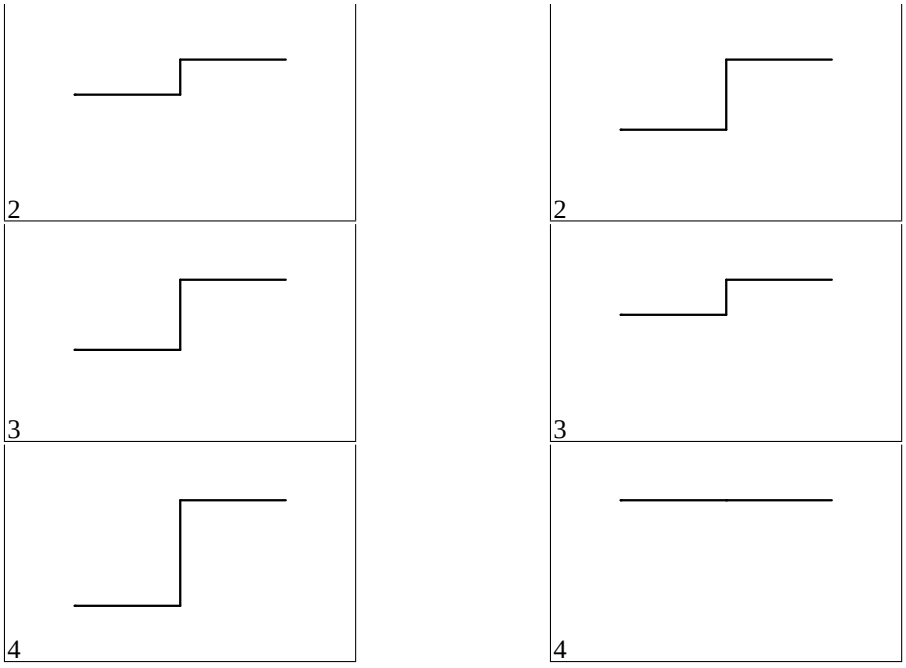


Figure 4.38: Insertion of degenerate segments. (a) shows the initial edge layout of two segments a and b. (b) inserts a degenerate segment, splitting the original segment a into two segments a and c, separated by the zero-length vertical segment b at $y = 50$. (c) inserts two degenerate segments at the edge bend between the original segments a and b.



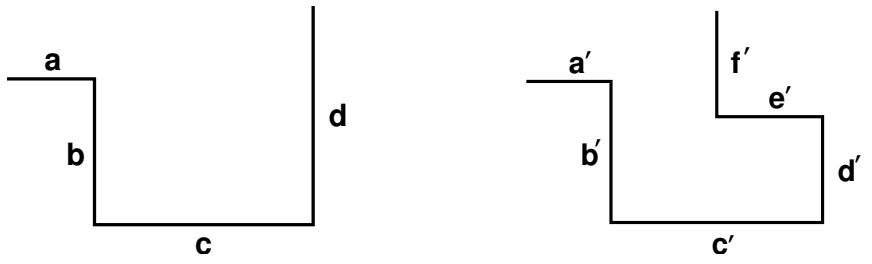
(a) inserting (b) removing

Animated Figure 4.39: Linear edge animation of inserting and removing an edge segment.

$0, 100, 0, 100, 100)$. Vertical degenerate segments have no clear N or S orientation, but may easily assume either, depending on how its adjacent segments change. Similarly for horizontal degenerate segments, which have no clear E or W orientation. As such, the orientation of a degenerate segment is represented by a dot '**.**'.

This technique can also be used to *normalise* all orthogonal edge layouts so that they start with an X segment and end with a Y segment (or vice-versa). This is simply done by inserting a degenerate segment at the start or end of the edge layout, where appropriate. Doing this simplifies the treatment of orthogonal edge layouts, thus, it is assumed hereinafter, that all orthogonal edge layouts have been normalised.

When the initial layout has fewer segments than the final layout, degenerate segments are added to the initial layout. The linear interpolation animation proceeds as before, and the visual effect is



(a) Initial (b) Final

Figure 4.40: Orthogonal edge animation problem instance showing insertion of an edge segment into the last segment.

13 14 15 16

Animated Figure 4.41: Linear animation of the orthogonal edge animation problem instance from Figure 4.40, inserting the degenerate segments at the leftmost endpoint.

the smooth insertion of two bends in the animation. In the other case, the final layout has fewer segments and has degenerate segments added, causing bends to be removed by the animation. Animated Figure 4.39 shows a simple example of these two cases.

The issue of exactly where to insert the degenerate segments remains. This is important because inserting degenerate segments away from their final location, can cause large sections of the edge to "shift". For example, Figure 4.40(a) shows an edge with four segments, and Figure 4.40(b) shows

this edge with an additional segment inserted mid-way in the last segment. Animated Figure 4.41 shows the linearly animated transition where the two additional segments are inserted at the start of the edge. Observe that the majority of segments appear to "slide" from the left to the right, and that segments a, b and c correspond to (or "transform" into) segments c', d' and e'. In addition, during the intermediate frames of the animation, the edge layout does not look similar to either the initial or final frame, despite the similarity between the initial and final frames. Animated Figures 4.42 and 4.43 show better animations of this transition, where segments a, b, c, d correspond to a', b', c', d'. We now present a heuristic method (without proof) for determining where to place inserted degenerate segments.

The problem has two parts. First, the position of the degenerate segments in the list of segments must be chosen, that is, each degenerate segment must be added between two existing segments. Second, the actual x or y coordinate value must be chosen for each degenerate segment.

In order to determine where in the list of segments the degenerate segments should be added, it is necessary to determine which segments correspond to one another in the initial and final layouts. This is achieved by using the Longest Common Subsequence (LCS) algorithm. The algorithm takes as input two strings, and returns the longest possible subsequence which is common to both strings. We use the orientation strings of the two layouts in question. For the example shown in Figure 4.40, the LCS algorithm is presented with the following two orientation strings:

E	S	E	N		and		E	S	E	N	W	N
a_1	a_2	a_3	a_4				b_1	b_2	b_3	b_4	b_5	b_6

giving two different longest common subsequences of length 4:

E	S	E	N		or		E	S	E	N
a_1	a_2	a_3	a_4				a_1	a_2	a_3	a_4
b_1	b_2	b_3	b_4				b_1	b_2	b_3	b_6

It is easy to determine that in the first case the segments b_5 and b_6 are the additional segments, whereas in the second case the segments b_4 and b_5 are the additional segments. Thus, the two possibilities are adding the two degenerate segments after a_4, or between a_3 and a_4:

E S E N . . and E S E . . N

If these degenerate segments are positioned at the corresponding bends in the edge layout, then the coordinates are determined by the neighbouring segments:

E	S	E	N	.	.		and		E	S	E	.	.	N
a_1	a_2	a_3	a_4	a_5	a_4				a_1	a_2	a_3	a_4	a_3	a_4

where a_5 is the y value of the endpoint of the edge, that is, where the edge meets the node at that endpoint. Animated Figure 4.42 shows the linear animation for each of these cases.

In fact, neither of these solutions are particularly desirable, because they both involve more motion than is necessary. This is the second part of the problem — choosing the x and y values for each degenerate segment that is to be inserted. In this case, it would be better to place the degenerate segments mid-way along the segment a_4, at precisely the location of b_5. This is achieved by first placing the degenerate segments at edge bends, as described above. Doing so causes degenerate segments to be characterised by identical neighbouring segments. Thus, the edge layout is searched for triplets of edge segments of the form α, β, α, which identifies the "central" segment (with value β) as a degenerate segment. The value of β is replaced by the value of the corresponding segment from the longer edge layout, thus moving the degenerate segment from the edge bend to the correct position along the edge segment. As such, it is easy to see that the final location of the degenerate segment is independant of the initial location. This is because the degenerate segment is initially placed at one of the two endpoints of the segment in question, and in both cases, it is detected and moved to the correct location along the segment.

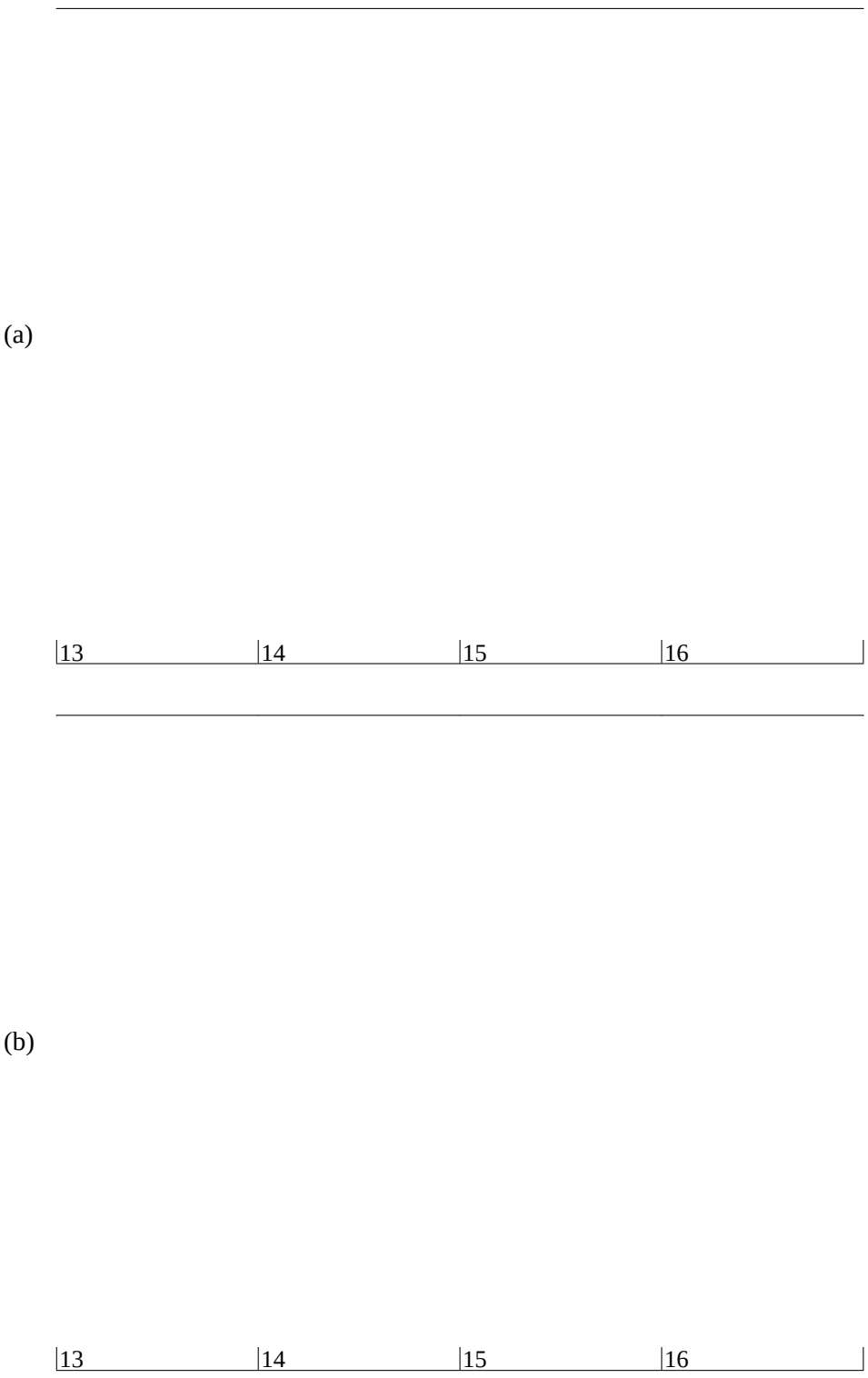
Returning to the example, doing this in either of the two cases shown in Animated Figure 4.42 gives

E	S	E	N	.	N
a_1	a_2	a_3	a_4	b_5	a_4

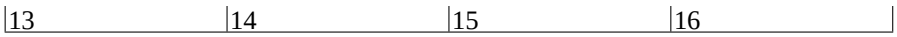
which is shown in Animated Figure 4.43. Clearly, this animation is superior to both of those shown in Animated Figure 4.42, as in this case there are half as many edges moving, and half as many edges "growing" (that is, changing length).

In fact, it is possible for there to be many solutions found by the LCS algorithm, not all of which are equivalent as is the case in Figure 4.42. This is particularly true of monotone sequences of segments, since the LCS algorithm cannot distinguish between these. Consider an edge with two segments, and inserting four segments of the same orientations, as shown in Figure 4.44. Now the LCS algorithm receives as input

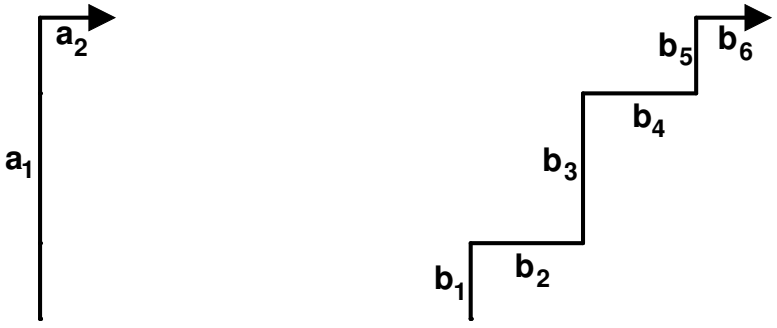
N	E		and		N	E	N	E	N	E
a_1	a_2				b_1	b_2	b_3	b_4	b_5	b_6



Animated Figure 4.42: Linear animation of the orthogonal edge animation problem instance from Figure 4.40, inserting the degenerate segments at either end of the last segment.



Animated Figure 4.43: Linear animation of the orthogonal edge animation problem instance from Figure 4.40, inserting the degenerate segments at the appropriate positions, based on the final layout.



(a) NE (b) NENENE

Figure 4.44: An orthogonal edge animation problem instance showing the insertion of four segments into one.

giving six solutions of length 2:

N	E	N	E	N	E
N	E	.	.	.	.
N	.	.	E	.	.
N	.	.	.	.	E
.	.	N	E	.	.
.	.	N	.	.	E
.	.	.	.	N	E

Which of these will be chosen is determined according to a type of distance squared based similarity metric for each layout. This metric indicates the level of similarity between the edge layout, with degeneracies added and the original edge layout, without degeneracies. The edge layout with the lowest metric value is chosen as the edge layout to use. How to compute the similarity metric for an edge layout is now described.

The final layout of the edge layout with fewer segments, including degeneracies, is compared to the initial layout of this edge layout. In order to do this, a list c is constructed for the edge layout with fewer segments, containing the elements c_i, for $0 \leq i \leq l-1$ (l the length of the LCS). This is a list of the indicies of the segments in the LCS. Now a list d is constructed for each LCS solution, which contains the elements d_i, the indicies of the segments in the LCS solution. The possible values for these elements are $0 \leq c_i \leq n-1$ and $0 \leq d_i \leq m-1$, where n is the number of segments in the initial edge layout and m is the number of segments in the final edge layout. In the above example, $l = 2$, the list c is given by

		c_0	c_1
N	E	0	1

and the list d is, for each potential final layout, given by

						d_0	d_1
N	E	.	.	.	.	0	1
N	.	.	E	.	.	0	3
N	.	.	.	.	E	0	5
.	.	N	E	.	.	2	3
.	.	N	.	.	E	2	5
.	.	.	.	N	E	4	5

The values of c and d are now converted to be the distance from either endpoint of the edge layout, that is, $c'_i = \min(c_i, n-1-c_i)$ and $d'_i = \min(d_i, m-1-d_i)$. The values for the initial layout are now given by the list c',

		c'_0	c'_1
N	E	0	0

which corresponds to the lists d' for the final layouts,

						d'_0	d'_1
N	E	.	.	.	.	0	1
N	.	.	E	.	.	0	2
N	.	.	.	.	E	0	0
.	.	N	E	.	.	2	2
.	.	N	.	.	E	2	0
.	.	.	.	N	E	1	0

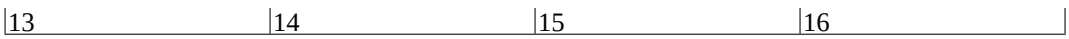
The similarity metric is given by

$$S = \sum_{i=0}^{l-1} u(c_i, d_i)[(c'_i - d'_i)^2 + 1]$$

where $u(c_i, d_i)$ is defined as 0 if the segments at indicies c_i and d_i have the same axis and value, and 1 otherwise.

This metric reflects that edge layouts which have common segments at similar relative positions are preferred. In addition, the relative position of segments is measured from either endpoint, reflecting that it is preferable to keep the endpoints of an edge stable and perform adjustments to the inner segments. The factor $u(c_i, d_i)$ reflects that segments with the same value do not change (visually), and so should not incur any penalty. The additional $+1$ term is included to give preference to layouts which have more segments with the same value, since the effect is to add to the metric the number of segments not ignored by $u(c_i, d_i)$.

In the example above, the metric values for each of the final edge layouts is:



Animated Figure 4.45: Linear animation of the orthogonal edge animation problem instance from Figure 4.44, inserting the degenerate segments at the appropriate positions, based on the final layout.

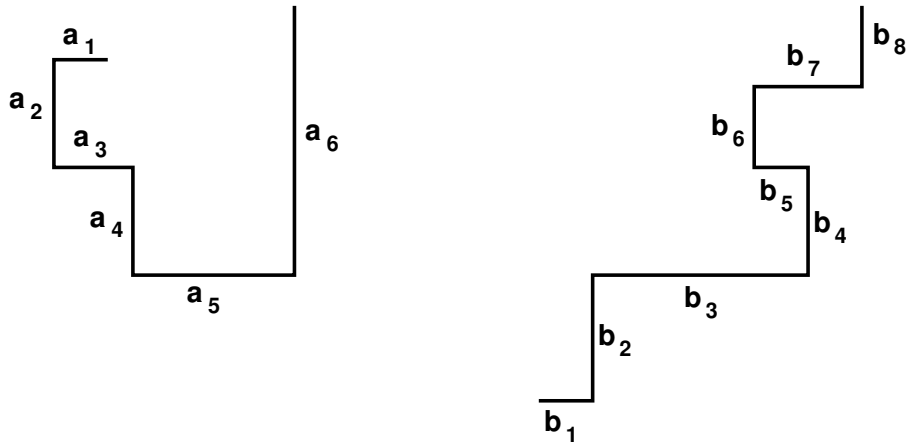
						S	
N	E	.	.	.	.	2	
N	.	.	E	.	.	5	
N	.	.	.	.	E	0	*
.	.	N	E	.	.	10	
.	.	N	.	.	E	5	
.	.	.	.	N	E	2	

The chosen layout is marked with an asterisk, and the linear animation transition corresponding to this choice is shown in Figure 4.45.

A non-trivial example of this algorithm is shown in Figure 4.46, which seeks to animate from

W	S	E	S	E	N		to		E	N	E	N	W	N	E	N
a_1	a_2	a_3	a_4	a_5	a_6				b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8

where the edge segments a_4, a_5 and a_6 have the same values as b_2, b_3 and b_6, respectively. This has two distinct longest common subsequences of length 3, “EEN” and “WEN”. The specific solutions in each of these two cases are considered separately, and their associated c_i, d_i and S values are



(a) WSESEN (b) ENENWNEN

Figure 4.46: A non-trivial orthogonal edge animation problem instance.

"EEN" case, Initial:

	W	S	E	S	E	N	c_0	c_1	c_2	c'_0	c'_1	c'_2	
	.	.	E	.	E	N	2	4	5	2	1	0	

"EEN" case, Final:

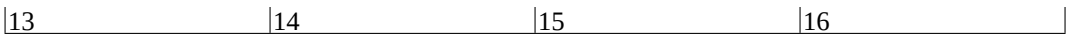
E	N	E	N	W	N	E	N	d_0	d_1	d_2	d'_0	d'_1	d'_2	S	
E	.	E	N	.	.	.	.	0	2	3	0	2	3	15	
E	.	E	.	.	N	.	.	0	2	5	0	2	2	5	
E	.	E	.	.	.	.	N	0	2	7	0	2	0	6	
E	.	.	.	.	.	E	N	0	6	7	0	1	0	7	
.	.	E	.	.	.	E	N	2	6	7	2	1	0	3	*

"WEN" case, Initial:

	W	S	E	S	E	N	c_0	c_1	c_2	c'_0	c'_1	c'_2	S
	W	.	E	.	.	N	0	2	5	0	2	0	13
	W	.	.	.	E	N	0	4	5	0	1	0	12

"WEN" case, Final:

E	N	E	N	W	N	E	N	d_0	d_1	d_2	d'_0	d'_1	d'_2	
.	.	.	.	W	.	E	N	4	6	7	3	1	0	



Animated Figure 4.47: Linear animation of the orthogonal edge animation problem instance from Figure 4.46.

As shown, the best value of S is the last solution in the "EEN" case, and the linear animation of this case is shown in Animated Figure 4.47.

4.3.4 Non-monotone edge layouts

This section deals with the animation of general edge layouts, that is, edge layouts that are not necessarily monotone. Simply applying the techniques from Section 4.3.2 to non-monotone layouts can easily cause ambiguities during the animation. As mentioned in Section 2.2.3, the edge routing

algorithm used is the Tom Sawyer Visualisation 6.0 software, which orthogonally routes the edges in the presence of the previously placed nodes. The edge routes found by this algorithm have the property of being shortest orthogonal paths between the nodes[2]. It has been shown that shortest orthogonal paths in the presence of rectangular barriers are always monotone in either x or y or both [30] (that is, the nodes of the inclusion tree layout are considered to be "barriers," around which the edges must be routed). As such, it is not necessary to consider the animation of non-monotone edges in this study. Nevertheless, for completeness we now present a simple heuristic algorithm for solving the Orthogonal Edge Layout Animation Problem for general (that is, non-monotone) edge layouts. This algorithm reduces the problem to the previously solved case of monotone edge layouts. However, as it is a heuristic, there are some cases for which it does not perform optimally, as described at the end of this section. Finding an efficient solution to this problem remains an interesting open problem. It consists of three broad stages:

Stage 1: Detection and animated removal of non-monotonicities in initial edge layout.

Stage 2: Linear animation of the resulting monotone edge layout.

Stage 3: Animated insertion of appropriate segments to reach the final (non-monotone) edge layout.

In fact, Stages 1 and 3 are effectively the same. This is because Stage 3 can be solved by applying the method for Stage 1 to the final layout and time-reversing the resulting animation. This has the effect of inserting the non-monotonicities, rather than removing them. Thus, the overall algorithm is actually:

Stage I: Obtain A' by removing all non-monotonicities from the initial layout A.

Stage II: Obtain B' by removing all non-monotonicities from the final layout B.

Stage III: Animate from A to A' (non-monotonicity removal).

Stage IV: Animate from A' to B' (monotone linear animation).

Stage V: Animate from B' to B (non-monotonicity insertion).

The linear animation in Stage 2 (Stage IV) merely animates the monotone orthogonal edge layouts as previously described. The method for Stage 1, the animated removal of non-monotonicities in an orthogonal edge layout, is now described.

[2]This may not necessarily be the case if constraints have been used to force edges to attach to particular sides of nodes, however, constraints have not been used in this study.

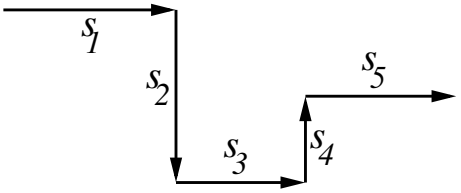


Figure 4.48: A simple non-monotonicity.

Removal of non-monotonicity has the following straightforward algorithm:

```
while E is not monotone do:
      Find a simple non-monotonicity
      Remove it
```

A non-monotone sequence of segments in an orthogonal edge layout is called a *non-monotonicity*. A non-monotonicity is *simple* if it contains no sub-sequences which are non-monotone, that is, if it contains no further non-monotonicities. Non-monotonicities which are not simple are called *compound*.

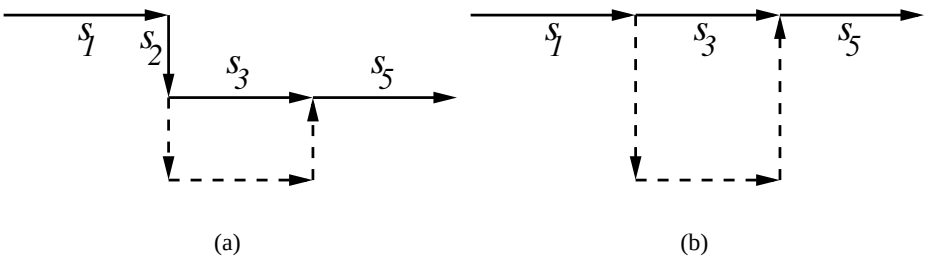
Testing for monotonicity (as in the condition of the while-loop) is simple. The method for locating a simple non-monotonicity is a linear search through the segments. The search looks for patterns of segments of the form:

E	S	E	N	E	or	N	E	N	W	N
s_1	s_2	s_3	s_4	s_5		s_1	s_2	s_3	s_4	s_5

More generally, s_2 and s_4 must be in opposite directions, s_1, s_3 and s_5 are in the same direction, and s_1 and s_5 may be absent (for example, at the start and end of the edge layout). (The search will generally consider s_2 to be the "current" segment, examining the other segments relative to it.) The form is shown in Figure 4.48. In addition, s_1 may be in the opposite direction to s_3, but only if the length of s_2, denoted by $|s_2|$ is greater than that of s_4, that is, $|s_2| > |s_4|$. Similarly, s_5 may be opposite to s_3 if $|s_4| > |s_2|$. It may never be the case that both s_1 and s_5 are opposite to s_3.

Once found, the simple non-monotonicity is removed. This is done by setting the value of s_3 to be the nearer of s_1 or s_5, that is,

$$s_3 = \begin{cases} s_1 & \text{if } |s_3 - s_1| < |s_3 - s_5| \\ s_5 & \text{if } |s_3 - s_1| \geq |s_3 - s_5| \end{cases}$$



(a) (b)

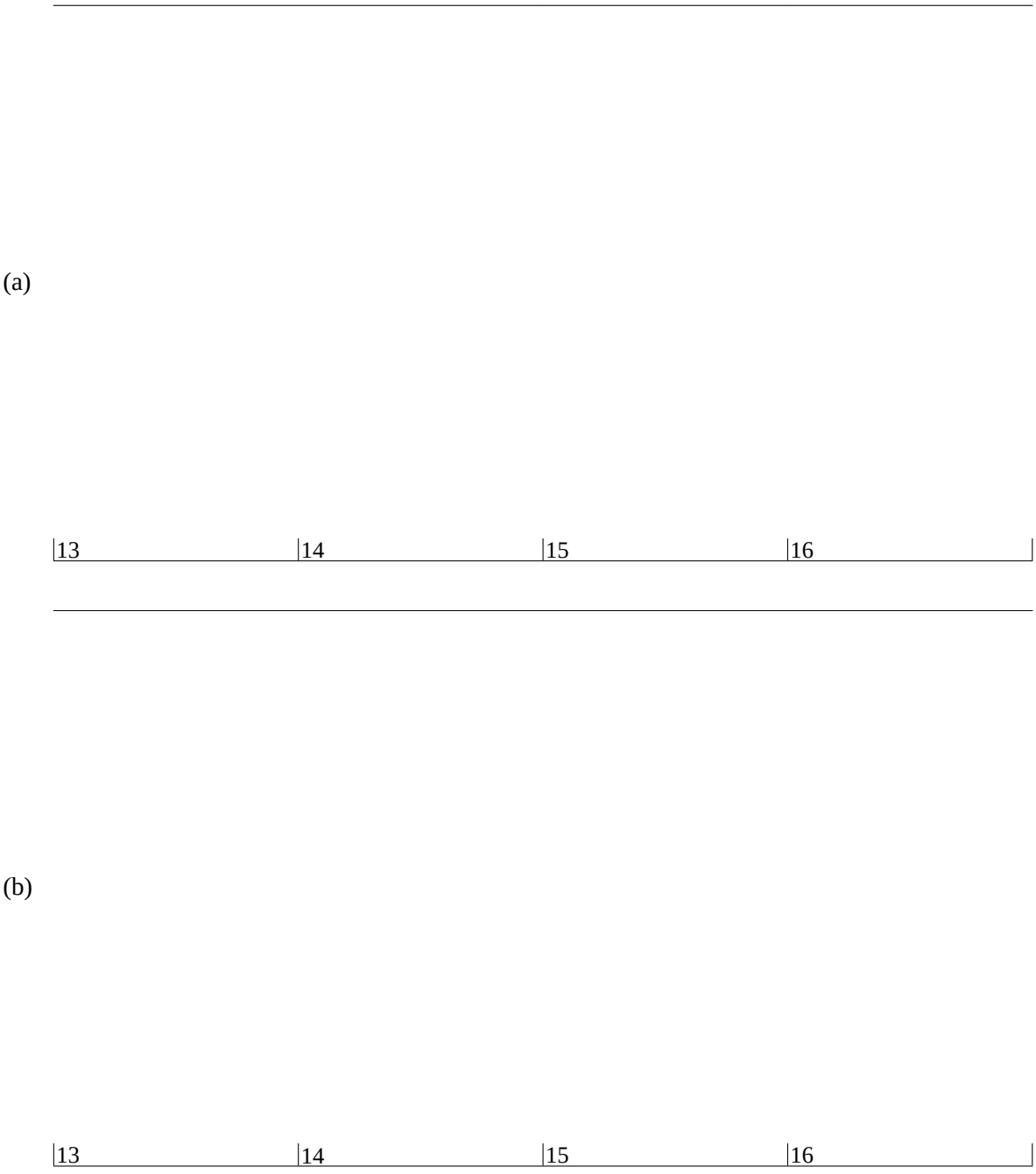
Figure 4.49: The result of removing the simple non-monotonicity from Figure 4.48. The previous layout is indicated by dashed lines.

(If either s_1 or s_5 are absent, then the value of the endpoint where the segment meets the node is considered in its place.) This causes s_2 or s_4 to become degenerate, allowing the removal of s_3 and either of s_2 or s_4 as appropriate. This is shown in Figure 4.49(a). If s_1 and s_5 have the same value, then when s_3 is set to that value both s_2 and s_4 will become degenerate, allowing the removal of s_2, s_3, s_4 and s_5. Figure 4.49(b) shows this special case.

This removal operation is quite simple to animate — simply linearly interpolate the value of s_3 from its initial value to the value of s_2 or s_4 as appropriate. An animation of this is shown in Animated Figure 4.50(a), with Animated Figure 4.50(b) showing the special case illustrated in Figure 4.49(b).

The output of the algorithm is correct, because the condition on the while-loop means that the result is a monotone orthogonal edge layout. The loop must terminate because, as will be shown, the non-monotonicity search finds a simple non-monotonicity if the edge layout is non-monotone; removal of a simple non-monotonicity causes a non-monotone sub-section of the edge layout to become monotone (clearly seen in Figure 4.49), thus the number of simple monotonicities present is reduced by 1.

That the search must find a simple non-monotonicity if the edge layout is not monotone is now proven. To do this, it is shown that in every compound non-monotonicity there exists at least one simple non-monotonicity, and that this is found by the search. By considering the entire non-monotone edge layout to be a compound non-monotonicity, it is seen that a simple non-monotonicity exists and must be found. If the edge layout is still not monotone after removing this simple non-monotonicity, then the process repeats until monotonicity is obtained, completing the proof of the algorithm.



Animated Figure 4.50: The orthogonal edge animation removing the simple non-monotonicity from Figure 4.49.

Theorem 4.3.5 *A compound non-monotonicity in a non-ambiguous edge layout must contain at least one simple non-monotonicity.*

First, assume that the compound non-monotonicity does not contain any degeneracies. This is because, by their very nature, degeneracies do not affect the visual appearance of the edge layout, and thus have no bearing on the monotonicity (or otherwise) of the edge layout. Any degeneracies present may easily be removed by an initial pre-processing stage.

Consider the vertical segments of the compound non-monotonicity. Since it is non-monotone, there must be at least one N segment and at least one S segment, giving it the form:

? . . . ? N ? . . . ? S ? . . . ? or

? . . . ? S ? . . . ? N ? . . . ?

These two forms are equivalent, so assume WLOG the former. The vertical segments between the N and S segments above must each be either N or S. Clearly, there must be at least one position between them that has the form ?N?S?, that is, $s_1 \mathrm{N} s_3 \mathrm{S} s_5$ using the notation in the simple non-monotonicity definition. (If all the intermediate vertical segments are N, then this occurs at the end, and if all the intermediate vertical segments are S, then this occurs at the beginning.) Now assume WLOG that $s_3 = \mathrm{E}$ (since the situation for $s_3 = \mathrm{W}$ is symmetric), that is, $s_1 \mathrm{NES} s_5$. This leaves three distinct cases:

Case 1: This is the case where the three remaining segments have the same orientation, that is, $s_1 = s_3 = s_5$, or ENESE. This trivially matches the criteria for being a simple monotonicity.

Case 2: This is the case where one of the three remaining segments has the opposite orientation, that is, $s_1 = s_3 = \overline{s_5}$ (ENESW) or $\overline{s_1} = s_3 = s_5$ (WNESE), where notationally $\overline{\mathrm{N}} = \mathrm{S}$, $\overline{\mathrm{S}} = \mathrm{N}$, $\overline{\mathrm{E}} = \mathrm{W}$, $\overline{\mathrm{W}} = \mathrm{E}$. Since both of these cases are identical by symmetry, assume WLOG the former, that is, ENESW. As stated in the definition of a simple non-monotonicity, this is simple if $|s_2| < |s_4|$. This leaves the case where $|s_2| \geq |s_4|$, as

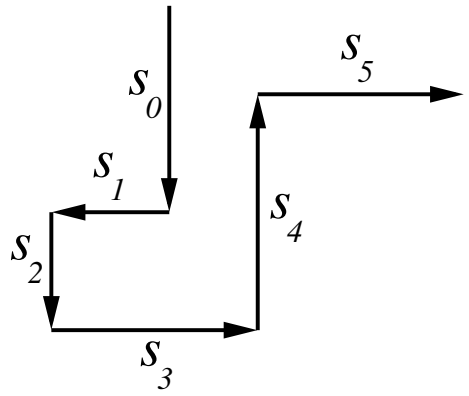
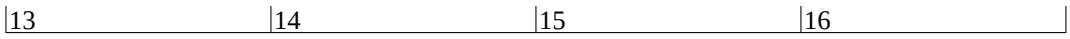


Figure 4.51: The case where $|s_2| \geq |s_4|$ in Case 2 of Theorem 4.3.5.

shown in Figure 4.51. Attempting to remove s_3 directly causes an ambiguity, where s_3 self-occludes s_1, as shown in Animated Figure 4.52. Since the edge layout is not ambiguous, simple geometry means that $|s_1| < |s_3|$, because $|s_2| < |s_4|$. Thus, in fact, the algorithm must previously have considered $s_0 s_1 s_2 s_3 s_4$, which is a non-monotonicity that must be simple (by case 2, since $s_0 = s_2 = \overline{s_4}$ and $|s_1| < |s_3|$). Animated Figure 4.53 shows the edge removal which actually takes place in this case.

Case 3: This is the case where s_1 and s_5 both have opposite orientations to s_3, that is, $\overline{s_1} = s_3 = \overline{s_5}$, or WNESW. This case is shown in Figure 4.54, and it is explicitly stated in the definition of simple non-monotonicities that this case is not simple. Removing s_3 directly causes an ambiguity where s_3 self-occludes s_1 if $|s_2| < |s_4|$, s_5 if $|s_2| > |s_4|$, and both s_1 and s_5 if $|s_2| = |s_4|$, as shown in Animated Figure 4.55. If s_0 is in the same direction as s_2, then the algorithm finds the simple non-monotonicity $s_0 s_1 s_2 s_3 s_4$, which satisfies case 2. If s_0 is opposite to s_2, then s_0 and s_4 are opposite to s_2, and it is possible to recursively consider the non-monotonicity $s_0 s_1 s_2 s_3 s_4$. This recursion continues until either s_0 is in the same direction as s_2, or the end of the edge is reached. In the former case, the non-monotonicity under consideration is clearly simple (as above), and in the latter, the non-monotonicity is simple because the absence of s_0 satisfies the simple non-monotonicity definition. Once this non-monotonicity has been removed, the previously considered one will also then be able to be



Animated Figure 4.52: Linear animation which attempts to directly remove s_3 from the case shown in Figure 4.51, causing self-occlusion between s_1 and s_3 in the last frame.

removed, by the same argument (that is, this recursive application of Case 3 will then "unwind"). The situation is trivially similar for the edge segments $s_2 s_3 s_4 s_5 s_6$. Animated Figure 4.56 shows how the example from Figure 4.54 is actually animated.

The situation when considering horizontal segments is identical to the treatment above of vertical segments. Thus, a non-ambiguous compound non-monotonicity must contain at least one simple non-monotonicity, which will be found by the algorithm.

Thus, this theorem shows that the algorithm presented is correct, and it will repeatedly detect and remove simple non-monotonicities from the edge layout until it is monotone.

13 14 15 16

Animated Figure 4.53: Linear animation which correctly removes s_2 from the case shown in Figure 4.51.

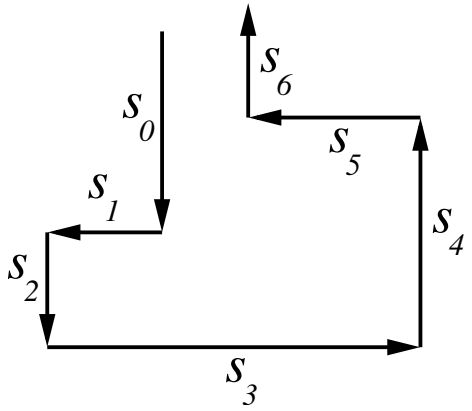
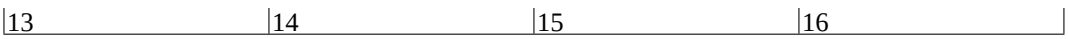


Figure 4.54: Case 3 of Theorem 4.3.5.



Animated Figure 4.55: Linear animation which attempts to directly remove s_3 from the case shown in Figure 4.54, causing self-occlusion between s_1 and s_3 in the last frame.

A pathological example of non-monotonicity is a "spiral", as shown in Figure 4.57. The only simple non-monotonicity is the inner-most one; removing it causes there to still only be one inner-most simple non-monotonicity. Animated Figure 4.58 shows the animation of the monotonisation of this edge layout.

This example makes it clear that the runtime performance of the algorithm is $O(n^2)$, for an edge layout with n segments. This is because in the worst case, it may be that the only simple non-monotonicity is at the end of the edge layout, requiring $O(n)$ segments to be examined to find (and remove) it. Up to $O(n)$ passes may be required through the edge layout (that is, there may be $O(n)$ simple non-monotonicites that the algorithm is required to remove), giving the overall performance of $O(n^2)$. It is easy to see that a lower bound on the performance of this edge monotonisation

13 14 15 16

Animated Figure 4.56: Linear animation which correctly removes s_2 from the case shown in Figure 4.54.

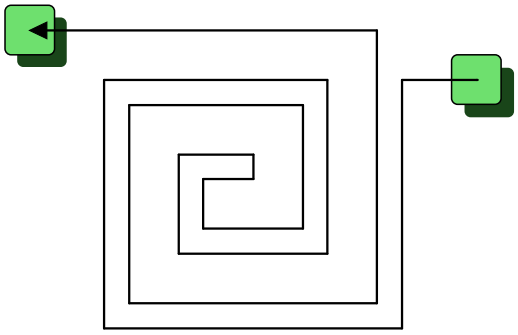
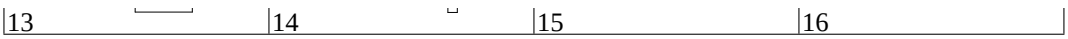
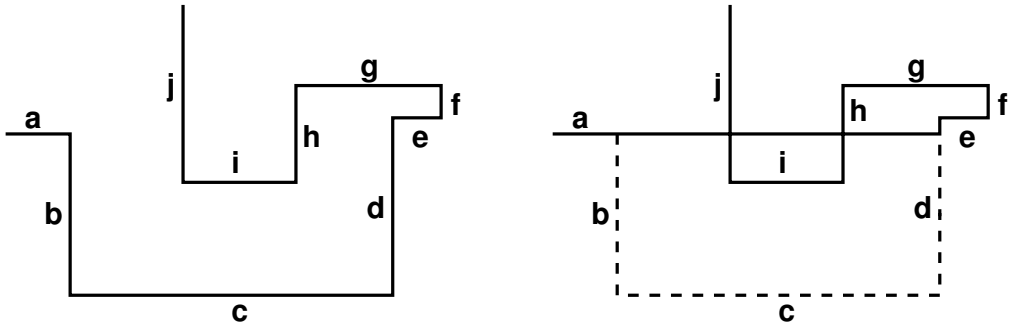


Figure 4.57: A spiral pathological case for removing non-monotonicities.



Animated Figure 4.58: Orthogonal edge animation showing the monotonisation of the pathological spiral from Figure 4.57.



(a) (b)

Figure 4.59: A problem associated with the removal of non-monotonicities in orthogonal edge layouts which are not monotone in x or y.

problem is $O(n)$, since each segment must at least be considered to be part of a non-monotonicity (lest an adversary change a segment that is not examined by a sub-linear-time algorithm). Although the performance of our algorithm is not close to this best case, it is adequate for our purposes because the value of n can reasonably be expected to be small, that is, $n \lesssim 10$. In fact, in the vast majority of instances, $n \leq 5$.

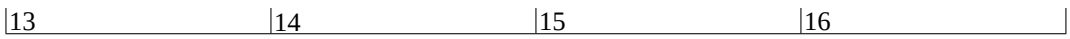
However, there are still some situations where self-intersection is not prevented by this simple algorithm. Consider the orthogonal edge layout shown in Figure 4.59(a). The segment sequence $abcde$ matches the criteria, and so segment c is removed, resulting in the orthogonal edge layout shown in Figure 4.59(b). A better sequence to consider is $defgh$ or $ghij$, removing f or i respectively. This situation only arises for edges which are not monotone in x or y. A thorough treatment of this is beyond the scope of this thesis (because, as described at the start of this section, non-monotone edges are not encountered in this study). One straightforward way of dealing with the problem would be to search the remaining segments for intersections with the segment being proposed for removal, although this is clearly inefficient. It may also be possible to prove the correctness of some sort of heuristic, for example, always removing the shortest possible segment first (which would remove f in the example above). Devising an efficient algorithm for removing non-monotonicities without creating self-intersections is left as an interesting open problem.

The original counter-example to linear interpolation of non-monotone orthogonal edge layouts, Figure 4.28, is shown animated using this algorithm in Animated Figure 4.60. This is clearly superior to the previously shown animations for it in Animated Figures 4.29 and 4.31.

4.3.5 Expanding and collapsing nodes

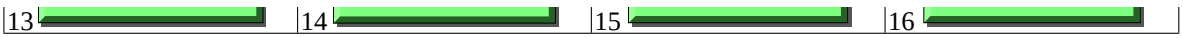
The final consideration, is the treatment of orthogonal edge layouts during the operations of expanding and collapsing cluster nodes. As previously, only the expanding of nodes is considered, with node collapsing being the obvious time-reversed animation. In this case, each meta-edge, adjacent to a node being expanded, is representative of one of more edges (which may also be meta-edges, or may be actual edges) that are to be revealed by the node expand operation. The Structural Zooming technique precomputes the layout after the node has been expanded, and identifies the *sub-edges* represented by each adjacent meta-edge. One *temporary edge layout* is constructed for each sub-edge, and the animation proceeds so that this temporary edge layout is animated from the initial meta-edge layout, to the particular sub-edge with which it is associated. Thus, the overall effect is of meta-edges "splitting" into several actual edges during the course of the animation. Similarly, when collapsing nodes, the sub-edges appear to "coalesce" into a single meta-edge. Animated Figure 4.61 shows an example animation of collapsing a node.

While this technique adequately deals with the meta-edges, it does leave the ends of the temporary edge layout "dangling" inside the expanding and collapsing nodes, that is, they are not directly incident on the boundary of any node. This is often an undesirable situation for a graph drawing,



Animated Figure 4.60: Orthogonal edge animation of the problem instance from Figure 4.28, animated using the edge animation strategy for non-monotone edge layouts.

however, it is adequate for our purposes. This is because the structure of the data is still maintained, since the edge endpoints are within the expanding (or collapsing) node (which has not yet completed the expand (or collapse) operation). For applications where this may be a problem, there are two possible solutions, both of which involve animating the child nodes in addition to the temporary edge layouts. Considering the expanding node case, in the first solution, the child nodes are initially the same size as their parent node, and their size and position simply animates linearly to their final position. (As always, the case of a collapsing node is simply the reverse of this.) This maintains the incidence of the temporary edge layouts against their respective child nodes, however, it has the disadvantage of significant occlusion between child nodes early in the animation. The second solution addresses this: a *temporary node layout* is created for each meta-edge incident to a child node. The initial size of this temporary node is given by the final size of the corresponding child



Animated Figure 4.61: An example orthogonal edge animation of meta-edges incident to an expanded node.

node, multiplied by the ratio of the initial to final size of the parent node[3]. The initial position of the temporary node is such that it is incident to the respective meta-edge, and the final position is that of the respective child node. Thus, while temporary child nodes that share a meta-edge will initially occlude one another, this occlusion will be short-lived as the nodes move toward their final positions. However, the disadvantage with this solution is that there is now also occlusion at the end of the animation, as the temporary nodes for a child node all converge on the same final position.

4.4 Evaluation framework

As described in Section 3.4, there are many possible ways of evaluating Structural Zooming. The application of Structural Zooming to relational data presented in this chapter has a large number of independant parameters, such as the layout algorithm, the tip-over and inclusion layout styles for trees, the parameters for the edge routing algorithm, the choice of data content measure and maximum detail threshold value. The “design space” formed by these parameters is evaluated with

[3] Although it is not possible to multiply two ‘sizes’ or take their ratio, here we are referring to these operations applied to both the width and the height.

an empirical investigation, running a series of experiments using a corpus of relational data input and navigation data input. Each experiment varies one parameter to be tested, and records the values of several different empirical measures for gauging the quality of the visualisation and animation for each operation performed.

As this evaluation is empirically based, the applicability of the results to real systems depends on two implicit assumptions:

Corpus validity requires the corpus of relational and navigation data to be indicative of those found in real systems.

Measure validity requires the empirical measures to be representative of the aspects that actually influence human understanding.

It is thought here that both of these assumptions hold. For corpus validity, care has been taken to ensure that the corpus data is sourced from a variety of real-world applications. This includes the navigation data, which is derived from both human and modelled sources. The application domains are Design Behaviour Trees, described in Chapter 5, Software Views, described in Chapter 6, and Citation Networks, described in Chapter 7. The corpus data in each application domain has been selected to ensure good coverage of the design space parameters.

For measure validity, we have used three fundamental categories to intuitively select formal empirical measures. These categories embody the possible ways in which the quality of an animation may be determined.

Static: The quality of each frame can be assessed individually, as though it were a static image and not part of an animation. This is justified by the fact that in an animation comprised of statically low quality frames, the quality of the overall animation itself is unlikely to be high. Static quality measures include all classical graph layout quality and aesthetic measures.

Endpoints: The initial and final frames of the animation largely determine the animation that must take place between them. Initial and final frames which are very different, will generally require large changes (that is, large amounts of movement) in order to animate between them. Similarly, initial and final frames which are similar will generally require only small changes during the animation. Since animations should be as simple as possible, the quality may

be considered to be inversely proportional to the difference between the initial and final frames. This may be expressed in general as $Q(n-1, 0)$, for a quality measure Q which compares the $(n-1)$-th (final) frame to the 0-th (initial) frame.

Dynamic: The final way in which an animation may be judged is by considering the "dynamic" changes between each pair of successive frames. This builds upon the previous category by evaluating the actual changes that occur during the animation. This is necessary because it is possible to have a poor animation (for example, one with much unnecessary movement) between very similar initial and final frames, with each intermediate frame having high static quality. Using the notation from the previous category, this may be expressed in general as

$$\sum_{i=1}^{n-1} Q(i, i-1)$$

The evaluation measures used draw heavily on existing principles from the field of information visualisation (for example, [23, 32, 45, 54, 55, 101, 125, 173]), the "minimalist" principles of good design (most notably, [159, 160, 161]) and the fields of perceptual and cognitive psychology (for example, [17, 99, 126, 166]). In particular, several of the measures are directly related to or inspired by Friedrich's empirical graph animation quality measures [54, 55]. The validation of these measures would be the subject of a formal user-study. However, due to the number of independant parameters this is a considerable task that falls outside the scope of this thesis. Instead, the evaluation measures are justified with referencing appropriate existing literature.

This section presents the empirical measures that are used in the experiments in Chapters 5, 6 and 7.

4.4.1 Minimise layout size ($\mathcal{M}_{\mathrm{MLS}}$)

This is a static measure which is an indication of the quality of the node layout produced. The *size* of a layout (sometimes referred to as the *compactness* or *resolution*), is an important graph drawing aesthetic [32, 125, 173]. This is because the detail-context tradeoff (discussed in Section 1.1) means that the smaller the layout size, the larger the detail is on a fixed size display.

The layout size is computed by taking the size of the window enclosing the layout, in the same

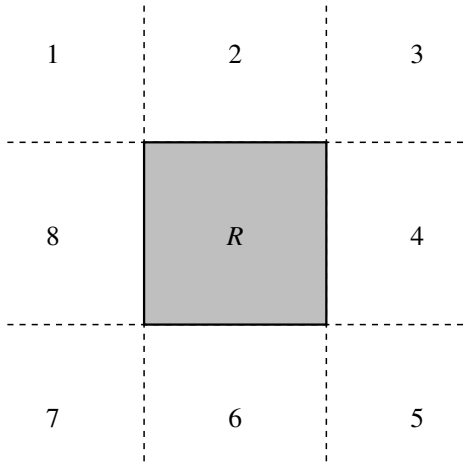


Figure 4.62: The eight orthogonal ordering regions surrounding node u, as numbered by the Orthogonal Ordering measure.

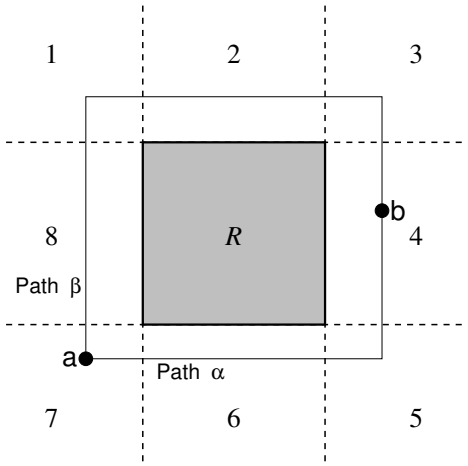


Figure 4.63: The difference between two orthogonal ordering regions is the smallest number of regions which must be traversed to go from one region to the other. Path α traverses 2 regions, while path β traverses 4 regions, thus $r_d(a, b) = 2$.

(arbitrary) units as the leaf node sizes (which are fixed). The window is the physical viewport used on the display to show the layout: defining the measure this way, rather than taking the size of the root node, takes into account blank space caused by a mismatch in display and root node aspect ratios. Note that when the display zooms in, we still take the size of the window, rather than the size of the root node or the top-level node being displayed. This gives an accurate indication of the level of on-screen detail shown, in consistent units (ie. the leaf node units).

The "size" of a rectangle is taken to be the perimeter of the enclosing window. Since the layout is fitted into the window, this is equivalent to the smallest enclosing rectangle of the display aspect ratio. As empirically shown in Section 2.3, this is considered to be the most stable and useful size measure for rectangles.

The measure is computed for the final frame of each animation.

4.4.2 Orthogonal ordering ($\mathcal{M}_{\mathrm{OO}}$)

This measure evaluates the changes in orthogonal ordering between the initial and final layouts of an orthogonal edge animation. As its name suggests, the orthogonal ordering considers the sorted order of nodes in the x and y axes. This has been found in previous studies to be strongly correlated with the user's mental map [23, 101]. As such, it is better if this ordering is preserved as much as

possible between the initial and final frames of the animation.

As in Section 4.2.1, the *orthogonal ordering regions* are defined to surround a rectangle, as shown in Figure 4.23. However, in this case the regions are numbered in a circular fashion, as shown in Figure 4.62. The *region number* of a point P with respect to a rectangle R is denoted by $r(R, P)$. For a node u, the bounding rectangle in the initial layout is $R(u)$ and in the final layout is $R'(u)$. The center point in the initial layout is $P(u)$ and in the final layout is $P'(u)$. The *region difference* between two orthogonal ordering regions, $r_d(r_1, r_2)$, is the smallest number of regions which must be traversed to go between the two regions.

$$r_d(r_1, r_2) = min((r_1 - r_2) \bmod 8, (r_2 - r_1) \bmod 8)$$

where r_1 and r_2 are region numbers. This is shown in Figure 4.63.

The orthogonal ordering difference of a set of nodes U is

$$d(U) = \sum_{u,v \in U} \sum r_d(r(R(u), P(v)), r(R'(u), P'(v))).$$

For the node-link layout convention, the measure is given by

$$\mathcal{M}_{\mathrm{OO}} = d(V),$$

where V is the set of all nodes. For the inclusion layout convention, the measure is given by

$$\mathcal{M}_{\mathrm{OO}} = \sum_{\substack{v \subset V \\ v \text{ expanded}}} d(C(v)),$$

where $C(v)$ is the set of child nodes of the expanded node v.

In all cases, off-screen nodes are completely ignored.

4.4.3 Minimise transient occlusions ($\mathcal{M}_{\mathrm{MTO}}$)

This measures the amount of overlap between nodes during the course of the animation. Nodes that occlude each other are almost always avoided by classical graph drawing algorithms [32], and when they are considered, significant effort is expended to remove the occlusions [101]. This measure is very similar to Friedrich's "Unnecessary Node Intersection" measure [54, 55]. It is

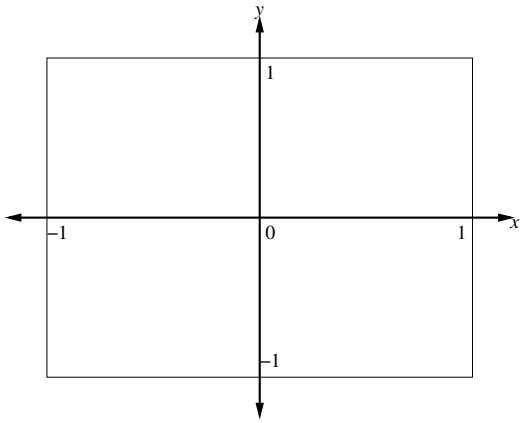


Figure 4.64: Normalised screen coordinates, $-1 \leq x \leq 1$, $-1 \leq y \leq 1$.

also similar to the "Temporary Edge Crossings" measure in that it considers transient problems during the animation. This measure is computed for each frame and then summed over the whole animation.

$$\mathcal{M}_{\mathrm{MTO}} = \sum_{\substack{u,v \in V \\ u > v}} \sum a(u,v) |R(u) \cap R(v) \cap R(S)|.$$

where $u > v$ indicates that each pair u, v should only be computed once (that is, it is an unordered pair), $R(u)$ and $R(v)$ are the regions occupied by those nodes, $R(S)$ is the region occupied by the screen, $|R|$ indicates the area of the region R, and $a(u, v)$ is defined as 0 if a descendant-ancestor relationship exists between u and v, and 1 otherwise.

4.4.4 Minimise motion ($\mathcal{M}_{\mathrm{MM}}$)

This is a measure of the amount of on-screen movement during an orthogonal edge animation. This is based on the idea that it is easier for humans to follow the motions of the nodes and edges when these motions are small (and therefore slow). As well as being intuitively sound, it is based on Friedrich's "Node Path Length" measure [54, 55], which reflects the idea that the distance travelled by nodes should be as short as possible. This measure considers the distance moved by nodes and edges between each pair of frames. It is given by

$$\mathcal{M}_{\mathrm{MM}} = \sum_{v \in V} d(P(v), P'(v)) \; + \; \sum_{e \in E, s \in S(e)} \sum |P(s) - P'(s)|,$$

where $d(p_1, p_2)$ is the Euclidean distance between p_1 and p_2, E is the set of all edges and $S(e)$ is the set of segments of edge e. It is assumed that, for each edge, the number of segments in the initial and final layouts is the same, as described in Section 4.3.3. The points $P(v)$, $P'(v)$, $P(s)$ and $P'(s)$ are first normalised to screen coordinates, as shown in Figure 4.64.

4.4.5 Minimise motion groups ($\mathcal{M}_{\mathrm{MMG}}$)

This measure is similar to the Minimise Motion measure, except that it is concerned with how the motion is grouped. This is because perceptual psychology research has shown that humans are able to combine the movements of distinct objects that have similar velocities [17, 34, 97, 99, 105], and are able to track up to four concurrent movements with little cognitive overhead [126]. Thus, a measure which gauges how well the movements of the various objects can be grouped into four, is sought.

Consider the collection of velocity vectors for each node and edge segment, computed as the Euclidean distance between the previous frame and the current frame, as in the Minimise Motion measure. This collection of vectors, each an ordered pair of real numbers, is then clustered in the plane using the Teitz-Bart (TB) heuristic for solving the k-medoids problem (also known as the p-median problem) [45, 156]. The number of clusters is set to 4, and the clustering process assigns each velocity vector to one of these clusters. The goal of the k-medoids problem is to minimise the quantity

$$M(C) = \sum_{i=0}^{n-1} d(p_i, \mathrm{rep}_C(c(p_i))),$$

where $P = \{p_0, p_1, ..., p_{n-1}\}$ is a collection of n points in the plane, $c(p)$ is the cluster point p belongs to, and rep is the *representative* point of the cluster, which must be an element of C. The set C is the collection of representative points, one per cluster, and must be a subset of P. Clearly, every point must be assigned to the cluster with the closest representative point. The clustering algorithm optimises the choice of C in order to minimise $M(C)$.

However, the value of $M(C)$ is sensitive to the initial random assignment of points to clusters and the order in which points are considered when updating the cluster representatives. If this is not considered, then the clustering obtained is not deterministic, and is therefore not useful in an emperical measure. As such, the seed of the random number generator used by the algorithm is fixed for each corpus instance, and the nodes of the instance are always considered in the same order. In this case, the quantity $M(C)$ is a good indication of the quality of the clustering [45, 75].

Thus, the value of $M(C)$ at the completion of the TB heuristic with four clusters is used as the value for the Minimise motion groups measure, and

$$\mathcal{M}_{\mathrm{MMG}} = M(C).$$

4.4.6 Preservation of topology ($\mathcal{M}_{\mathrm{POT}}$)

This measure considers the embedding of the clustered graph in the plane, that is, the edge crossings which occur, and the faces created by nodes and crossings, collectively referred to as 'the topology'. Edge crossings have been found to be a very important graph drawing aesthetic [32, 125], in that minimising the edge crossings helps to maximise the user's understanding of the graph. However, this measure is not concerned with minimisation of crossings. Rather, it is concerned with the minimisation of *changes* in topology, or the *preservation* of the topology, and so the measure incurs a positive penalty whenever the topology changes. In this way, it is similar to Friedrich's "Temporary Edge Crossings" measure [54, 55], which is concerned with edge crossings induced during the course of the animation. It is also related to the "Constant Edge Length" measure, in that it seeks to preserve a property throughout the animation. Previous work has investigated the use of constraints (both user-specified and system-generated) to preserve properties such as the topology when changing a graph drawing [23]. The measure consists of two sections, incident edge ordering around each node, and edge crossings.

The edge ordering section examines each node and sorts the incident edges according to their order around the boundary of the node. This is used to compute the edge order in the current layout as a permutation of the edge order in the previous layout. This is represented as a permutation of the integers $\{1, 2, ..., n\}$, denoted by p_i, where $1 \leq i \leq n$ and $1 \leq p_i \leq n$, for n incident edges. We define the *cyclic distance* between two permutations q and r as

$$d(q, r) = \min_{k=0}^{n-1} \left(\sum_{i=1}^{n} (q_i - r_{(i+k) \bmod n})^2 \right),$$

that is, the minimum sum of the squared differences between the elements of the permutations, over all possible cyclic rotations of one of the permutations. In this case, the interest lies in $d(I, p)$, where $I_i = i$ is the identity permutation. Thus, the edge ordering part of this measure is given by

$$\mathcal{M}_{\mathrm{POT_a}} = \sum_{v \in V} d(I, p(v)),$$

where $p(v)$ indicates the permutation in edge order for node v.

The edge crossings section of the measure considers edges and segments which cross. Each pair of segments which cross is called a *segment crossing*. Clearly, a pair of segments may either cross once or not at all. The *segment crossing orientation* is the concatenation of the orientations of the segments that cross. Since crossing segments cannot both have the same axis, the y segment orientation is listed first, giving the set of possible segment crossing orientations $B = \{\mathrm{NE}, \mathrm{NW}, \mathrm{SE}, \mathrm{SW}\}$.

The total number of segment crossings between edges e_1 and e_2 in the previous layout is denoted by $n(e_1, e_2)$, and in the current layout by $n'(e_1, e_2)$. This is 0 if edges e_1 and e_2 are distinct and do not cross. The number of segment crossings of orientation b in the previous layout is denoted by $n_b(e_1, e_2)$, and in the current layout by $n'_b(e_1, e_2)$, where $b \in B$.

The number of *unchanged* segment crossings between edges e_1 and e_2 is given by

$$N_u(e_1, e_2) = \sum_{b \in B} \min\big(n_b(e_1, e_2), n'_b(e_1, e_2)\big)$$

The number of *changed* segment crossings is given by

$$\begin{aligned} N_c(e_1, e_2) &= \min\big(n(e_1, e_2), n'(e_1, e_2)\big) - n_u(e_1, e_2) + \big|n(e_1, e_2) - n'(e_1, e_2)\big| \\ &= \max\big(n(e_1, e_2), n'(e_1, e_2)\big) - n_u(e_1, e_2) \end{aligned}$$

The first term is the smaller number of segment crossings, minus the number of unchanged segment crossings — that is, the number of segment crossings which have changed. The second term is the difference in the number of segment crossings — that is, the number of segment crossings which have been added (or removed, as appropriate).

Now the value of the measure is the square of the number of changed segment crossings, summed over all pairs of edges.

$$\mathcal{M}_{\mathrm{POT_b}} = \sum_{e_1, e_2 \in E} \sum N_c(e_1, e_2)^2$$

Finally, the overall measure is simply the addition of these two sections

$$\mathcal{M}_{\mathrm{POT}} = \mathcal{M}_{\mathrm{POT_a}} + \mathcal{M}_{\mathrm{POT_b}}$$

4.4.7 Preservation of bends ($\mathcal{M}_{\mathrm{POB}}$)

This is similar to the Preservation of Topology ($\mathcal{M}_{\mathrm{POT}}$) measure, except that it is concerned with the *preservation* of bends, rather than their minimisation. For orthogonal drawings, bends are also linked to user understanding, but not as strongly as crossings [32, 125], and as such, their preservation during the animation is also expected to be beneficial. However, the removal of an orthogonal edge clearly reduces the visual complexity of the visualisation (since three consecutive segments are smoothly reduced to one, by the "middle" segment shrinking to zero length). Thus, this measure introduces the relaxation that removing a bend is not as bad as inserting a bend. It examines all corresponding pairs of edge segments between the previous layout and the current layout. If the orientations are the same, then the segment has not changed orientation, and so no penalty is incurred. If the orientation of the segment in the current layout is '**.**' then a segment has been removed, and a penalty of 1 is incurred. If the orientation of the segment in the previous layout is '**.**' then a segment has been inserted, and a penalty of 2 is incurred. Otherwise, the segments must be opposite, that is, N and S, or E and W, indicating that the segment has "flipped". This corresponds to removing and then re-inserting the segment, however, as it happens in a short space of time, is it known that the segment is not being removed permanently and will return immediately in the same position, thus, a penalty of 2 (rather than 3) is incurred.

The measure is given as

$$\mathcal{M}_{\mathrm{POB}} = \sum_{e \in E, s = S(e)} \sum f\big(b(s), b'(s)\big)$$

where $S(e)$ is the set of segments in edge e, $b(s)$ is the orientation of s in the previous layout and $b'(s)$ the orientation in the current layout. The function f indicates the penalty associated with particular segment orientation changes, and is defined as

$$f\big(b, b'\big) = \begin{cases} 0 \text{ if } b = b' \\ 1 \text{ if } b' = \text{.} \\ 2 \text{ if } b = \text{.} \\ 2 \text{ if } b = \overline{b'} \end{cases}$$

where $\overline{\mathrm{N}} = \mathrm{S}$, $\overline{\mathrm{S}} = \mathrm{N}$, $\overline{\mathrm{E}} = \mathrm{W}$, and $\overline{\mathrm{W}} = \mathrm{E}$.

4.5 Navigation strategies

As described in Section 3.4, "navigation strategies" are a hybrid method of obtaining navigation paths through each corpus data file. The navigation strategies are models for testing, devised intuitively after examining real navigations through the data. They are designed to be used for comparing the independant parameters in the application of Structural Zooming to relational data, under conditions that are sufficiently similar to real-world navigations.

Navigation strategies operate by repeatedly selecting nodes to expand. If the selected node is inside the current zoomed-in display, it is expanded, otherwise, the display is zoomed out by a single level. Generally, navigation strategies run until their natural completion, but they may also be set to run until a predetermined number of nodes have been visited.

4.5.1 Target node - perfect

This navigation strategy pre-selects a single distinguished node known as the *target node*. This is the node which the hypothetical user is searching for. This strategy always selects the collapsed node which is an ancestor of the target node. This continues until the target node is displayed. Note that the target node is usually a leaf node, although it need not be.

4.5.2 Target node - normal

This navigation strategy is similar to the "perfect" target node strategy, except that the choice of node is probabalistic — there is a certain chance of selecting the correct collapsed node, and a certain chance of making a "mistake". The particular *mistake model* used is as follows. There is a $2/3$ chance of immediately selecting the correct node. In the remaining $1/3$, the collapsed nodes are arranged into an array and sorted by their graph theoretic distance from the correct node. An index into this array is selected using a Gaussian normal distribution, scaled such that 3 standard deviations (99.7% confidence interval) is at the end of the array[4]. Thus, the chance of choosing a node far away from the correct one decreases exponentially, and there is still a significant chance of selecting the correct node, compared to other nodes.

[4] In the rare case of an index past the end of the array being selected, the last element of the array is used.

4.5.3 Random

This navigation strategy is simply a random walk through the clustered graph. For each operation, a collapsed node is selected from the clustered graph at random. This continues while there are collapsed nodes to choose from.

4.6 Conclusion

This chapter has presented the application of Structural Zooming (described in Chapter 3), to the relational data types of trees and clustered graphs (described in Chapter 2). First, the Structural Zooming technique was presented for h-v trees in the inclusion and node-link styles. This involved the definition of detail increasing and decreasing operations as the opening and closing of cluster nodes and level zooming in and out. Animations were presented for these operations, as well as for changing the layout from horizontal to vertical and vice-versa. The Structural Zooming technique was applied to arbitrary inclusion layouts with the Stable Jewellery Box Inclusion Layout Algorithm to minimise layout changes. It was extended to clustered graphs by considering the animation of orthogonal edge layouts in arbitrary inclusion tree layouts. Empirical measures were presented for assessing the quality of the transitions in the Structural Zooming of relational data. Finally, some navigation strategies were presented for modelling user navigation through tree and clustered graph Structural Zooming.

CHAPTER 5

Design Behaviour Trees

The first empirical case study of Structural Zooming is that of *Design Behaviour Trees* (*DBTs*). Section 5.1 introduces and describes DBTs as they are used in this investigation. Section 5.2 describes the data source for the investigation, how the data is represented within the Structural Zooming system, and sets out the experimental design of the empirical investigation. Finally, Section 5.3 presents the results. The conclusions derived from this investigation are analysed and interpreted in Chapter 8, along with those presented in Chapters 6 and 7.

5.1 Background

Design behaviour trees (*DBT*s) provide representations of behavioural requirements in software engineering, developed by Dromey [36, 37]. They are constructed by composing the behaviour trees for each functional unit of a software system, according to a set of well-defined rules. They effectively allow the software system to be built directly from its functional requirements, rather than the more traditional activity of building a system that happens to satisfy those requirements. Each transformational step, from the natural language requirements specification document through to skeletal source code for the system design, is deliberately constructed to be deterministic, auditable and expose design flaws as soon and clearly as possible. Design behavour trees are an important intermediate stage in this process.

The motivation behind such research is clear — software engineering has a history of being particularly poor in terms of its ability to effectively design and develop “quality” software, that is to say, software low in defects, security flaws, inefficiencies, and the like. The cost of fixing a design flaw increases considerably as the software development lifecycle proceeds. Many projects exceed time or budget constraints (or both) as a result of fixing or re-engineering design issues that ought to have been addressed before coding had begun.

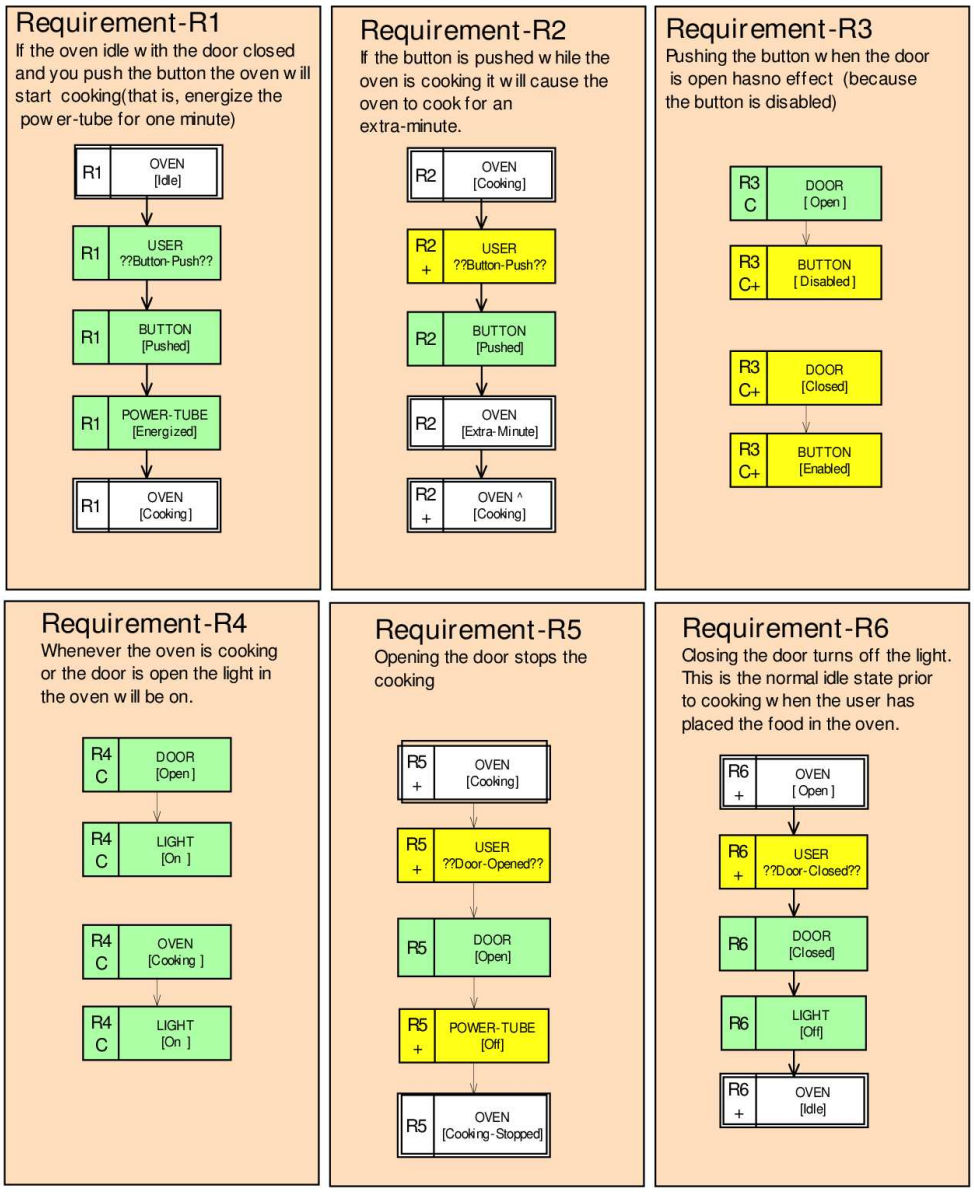


Figure 5.1: A set of 6 simple requirements for a microwave oven and the corresponding DBTs. Courtesy of Geoff Dromey [35].

DBTs are created by a two stage process. The first is to translate each functional requirement into a simple, straightforward, path of requirements. Figure 5.1 shows a set of six simple requirements for a microwave oven and the corresponding DBTs. Each node may represent a component, state or event. The second stage is the *integration* (or composition) of the individual requirement DBTs into a single overall DBT. Integration is achieved by merging the trees at common nodes, as shown in Figure 5.2. Figure 5.3 shows the overall DBT for the microwave oven system.

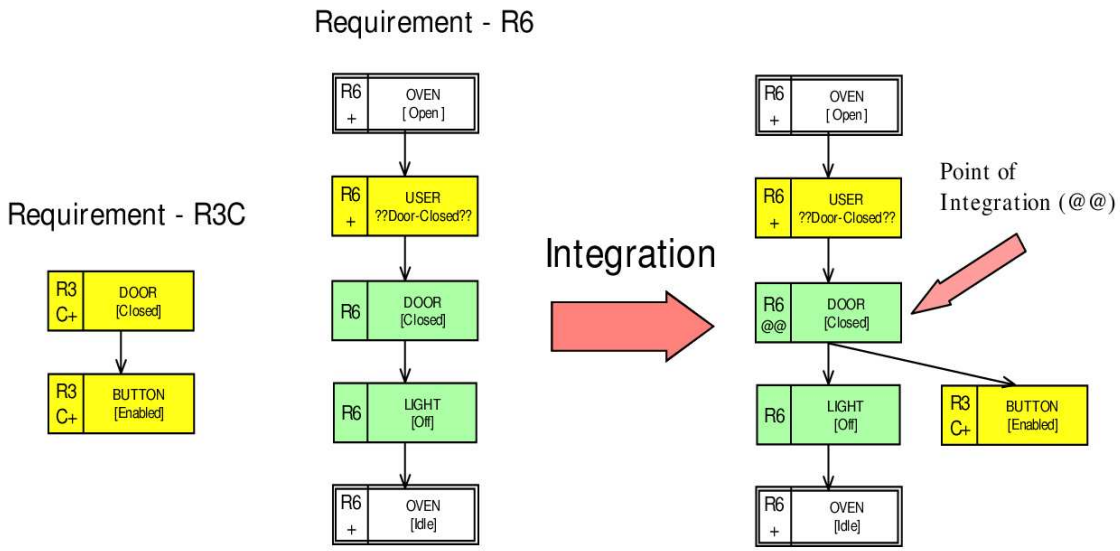


Figure 5.2: The result of integrating R3C and R6 from Figure 5.1. Courtesy of Geoff Dromey [35].

At first, DBTs were constructed and drawn by hand. However, as the size of the software systems studied using DBTs has increased, this became a limitation in their use. Some of the largest systems include up to 500 requirements and over 1000 unique nodes in the DBT. Individually placing each of these nodes is tedious and time-consuming, more importantly, displaying such large trees is difficult on computer displays, causing researchers to resort to large-format paper displays [35]. For these reasons, an interactive Focus + Context system, such as Structural Zooming, may be a useful technique for the interactive visualisation of these large trees on computer displays.

5.2 Experimental design

DBTs are naturally represented as node-link or inclusion style trees with Structural Zooming. For simplicity, much of the "visual syntax" present in Figures 5.1, 5.2 and 5.3 (such as double-boxes and colour) are omitted, and all types of nodes are considered equally. The data is obtained from DBT researchers at Griffith University, and converted into machine-readable trees using a semi-automatic conversion process.

The corpus of files used in this investigation, is the same as used in Section 2.3, including the non-DBT ontology files. Table 5.1 lists the sizes of these files, and Figure 5.4 shows the Enterprise-Ontology file in node-link and inclusion layout styles.

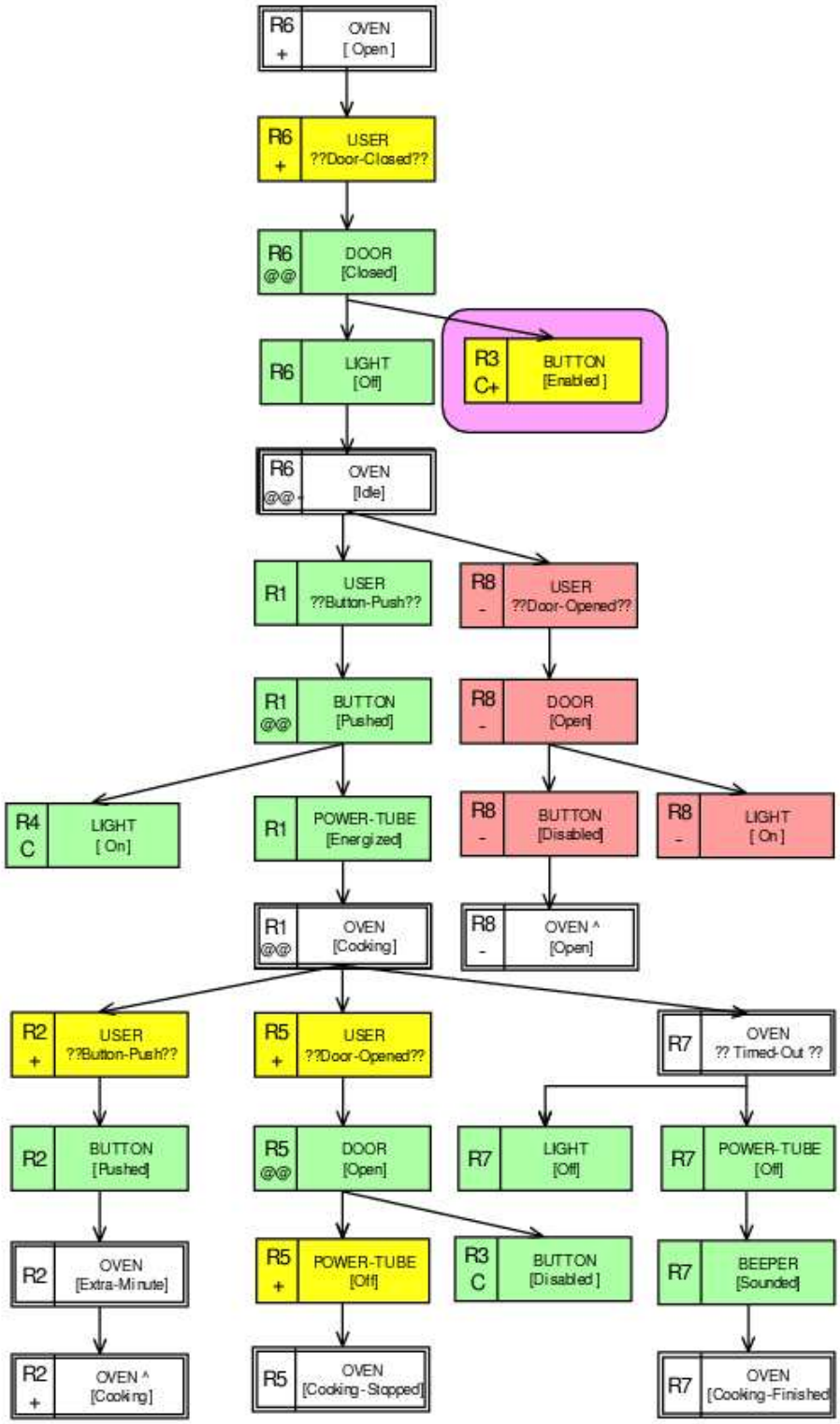
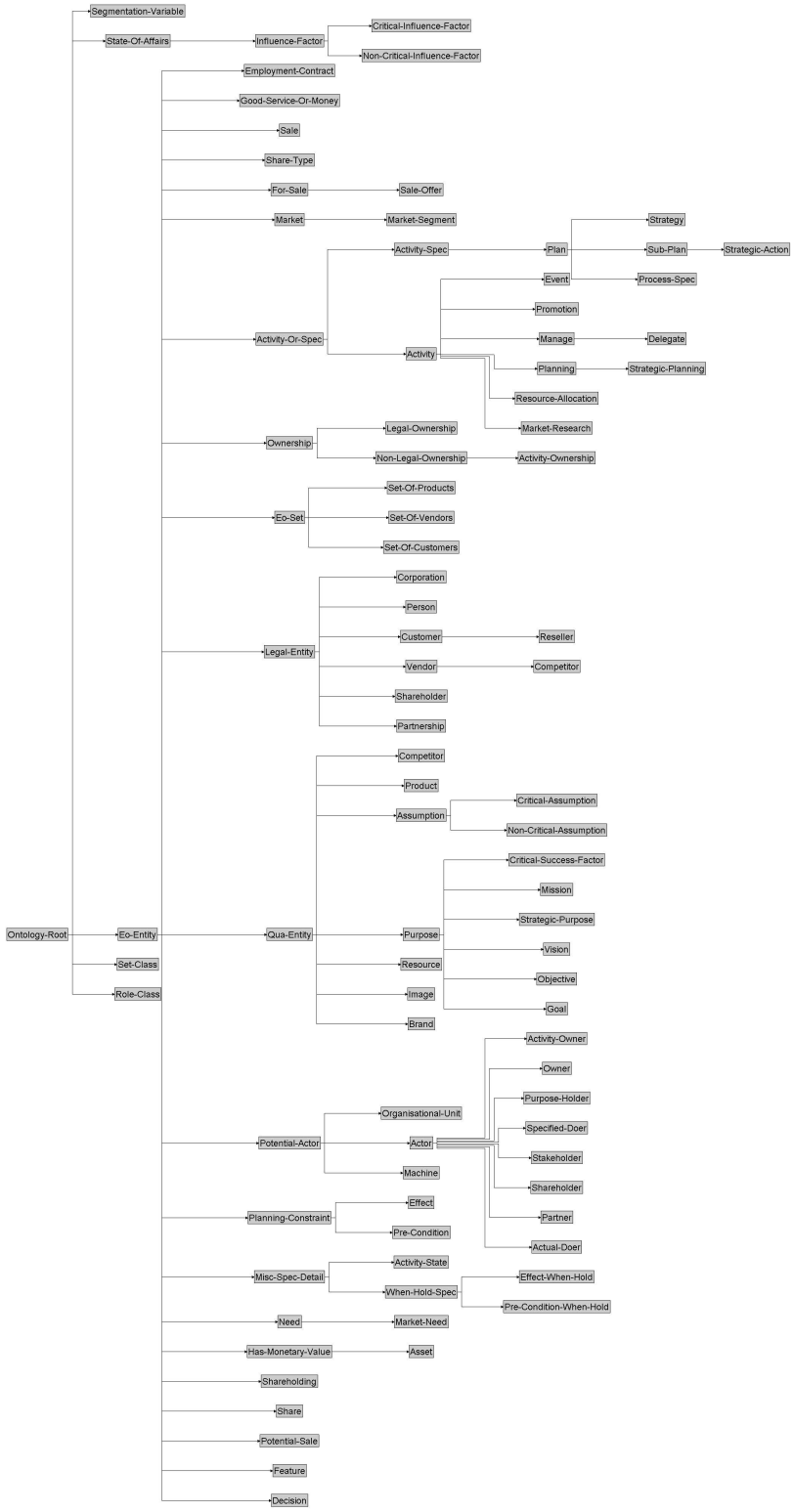
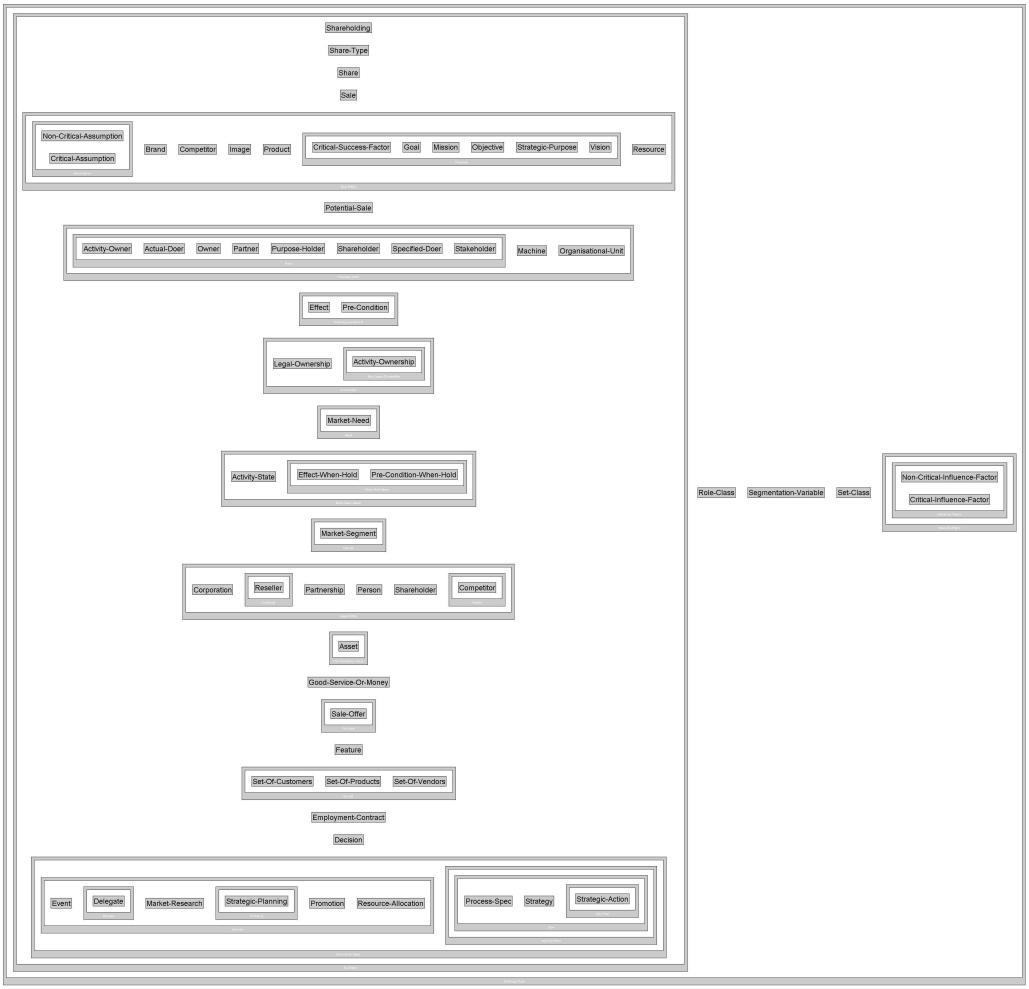


Figure 5.3: The complete integration of all the requirements for the microwave oven system. Courtesy of Geoff Dromey [35].



(a) Node-link Layout Style

Figure 5.4: The Enterprise-Ontology class hierarchy.



(b) Inclusion Layout Style (Minimum Enclosing Square)

Figure 5.4: The Enterprise-Ontology class hierarchy.

Navigation paths were sourced from both logfiles and navigation strategies. For each data file, two logfiles were obtained from software quality researchers at Griffith University who were familiar with the structure of DBTs [35]. The “target node — perfect” and “target node — normal” navigation strategies were used. For each, two sets of target nodes were created, being random samples of 25% and 75% of the leaf nodes. The motivation behind these differing sample sizes, is that the 25% sample size corresponds to a small exploration of the data file, such as a search for a small number of targets, whereas the 75% sample size corresponds to a larger and more thorough exploration of the data file. This gives six options[1] of navigation path for each data file, being:

[1] In fact, not every data file has exactly six navigation paths, as some files have one or three logfiles.

Tree	Number of Nodes
Integrated Low Level DBT	504
Multi-Sensory Taxonomy	290
Satellite Low Level DBT	269
Enterprise Ontology	95
Integrated High Level DBT	89
Mine Pump DBT	78
Online Shopping Low Lvl DBT	43
Online Shopping Med Lvl DBT	40
12207 Acquisition DBT	40
Satellite High Level DBT	33
Car System DBT	22
Online Shopping High Lvl DBT	21
Man Fishing DBT	17

Table 5.1: Sizes of the input trees shown in decreasing order.

1. Logfile 1
2. Logfile 2
3. Target node — perfect, 25% sample size
4. Target node — perfect, 75% sample size
5. Target node — normal, 25% sample size
6. Target node — normal, 75% sample size

Four inclusion tree layout algorithms are used.

- The greedy h-v inclusion layout algorithm, presented in Section 2.2.1.
- The optimal h-v inclusion layout algorithm, presented in Section 2.2.1.
- The jewellery box inclusion layout algorithm, presented in Section 2.2.2.
- The stable jewellery box inclusion layout algorithm, presented in Section 4.2.1.

For the h-v inclusion layout algorithms, the three possible h-v node rotation strategies are considered, as described in Section 4.1.2.

- Linear rotation.

- Circular rotation.
- Orthogonal rotation.

Additionally, the node-link tip-over layout style is considered for the two h-v layout algorithms (greedy and optimal). In this case, only orthogonal node rotation is considered. In total, this gives 10 possible combinations of layout style and algorithm:

	Style	Algorithm	Rotation
1	Inclusion	Greedy h-v	Circular
2	Inclusion	Greedy h-v	Linear
3	Inclusion	Greedy h-v	Orthogonal
4	Inclusion	Optimal h-v	Circular
5	Inclusion	Optimal h-v	Linear
6	Inclusion	Optimal h-v	Orthogonal
7	Node-link	Greedy h-v	Orthogonal
8	Node-link	Optimal h-v	Orthogonal
9	Inclusion	Jewellery Box	
10	Inclusion	Stable Jewellery Box	

Three possible choices are considered for the detail-reducing threshold, as described in Sections 3.3 and 4.1.1.

- $N(12)$ — At most 12 on-screen nodes.
- $N(24)$ — At most 24 on-screen nodes.
- $N(36)$ — At most 36 on-screen nodes.

The logfile navigation paths use only $N(12)$, because this is the detail-reducing threshold that was used when the logfiles were obtained. In general, it is not possible to use a different detail-reducing threshold with a logfile, as doing so would cause the set of on-screen collapsed nodes to be different to that available to the user creating the logfile.
This is an investigation using trees, so the following set of measures, described in Section 4.4, is used.

- $\mathcal{M}_{\mathrm{MM}}$ — Minimise Motion

- $\mathcal{M}_{\mathrm{MMG}}$ — Minimise Motion Groups
- $\mathcal{M}_{\mathrm{MLS}}$ — Minimise Layout Size
- $\mathcal{M}_{\mathrm{MTO}}$ — Minimise Transient Occlusions
- $\mathcal{M}_{\mathrm{OO}}$ — Orthogonal Ordering

The Preservation of Bends ($\mathcal{M}_{\mathrm{POB}}$) and Preservation of Topology ($\mathcal{M}_{\mathrm{POT}}$) measures are not used, since they only refer to orthogonal edge layouts which are only present in clustered graphs, not trees.

5.3 Results

The results are presented according to each of the individual parameters that were varied, aggregated over the corpus of data files and navigations. Full results are available on the CD accompanying this thesis, as described in Appendix A. A discussion of the results is presented in Chapter 8.

Results are presented in two forms: a set of "full" plots of the sorted values of each measure, and a set of three "summary" plots showing the maximum value, sum and average value, for each measure. The full plots show the value of the measure for each user operation, be it recorded in a logfile or generated by a navigation strategy. These user operations have been sorted according to the measure value, and the values joined in a line plot (that is, the x axis does not correspond to time, or to the order in which the operations were performed). The values in the summary plots are independently normalised for each measure. In general, the summary plots bring out the salient aspects of the parameter being presented, without overly simplifying the data. The nature of the investigation, particularly as shown by the sorted value plots, suggests that histograms would be a superior way to summarise and present the results: Section 6.3 shows that this is not the case.

Some comparisons do not contain the same number of operations. This is due to differences in the parameters causing differences in the number of operations; for example, a random sample size of 25% has less target nodes than a random sample size of 75%, and therefore also has less operations. In these cases, the maximum value and average value summary plots are more useful than the remaining plots.

5.3.1 Layout algorithm — Inclusion style

Figures 5.5 to 5.9 show the sorted results for each quality measure against the choice of layout algorithm in the inclusion style. Figures 5.10 to 5.12 show the maximum value, sum and average value summaries of the results.

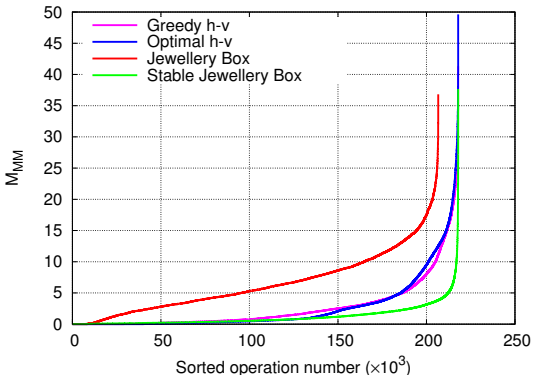


Figure 5.5: Inclusion layout algorithm comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

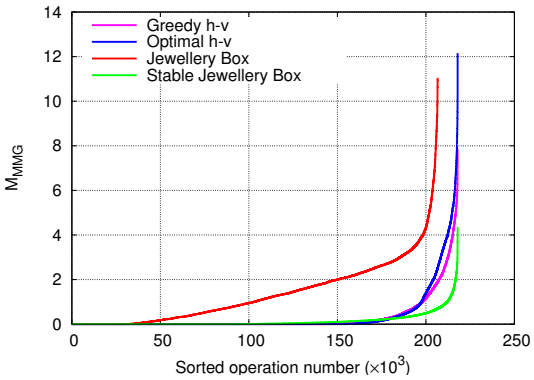


Figure 5.6: Inclusion layout algorithm comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

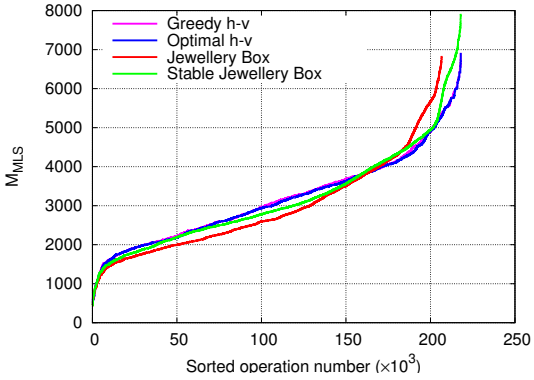


Figure 5.7: Inclusion layout algorithm comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

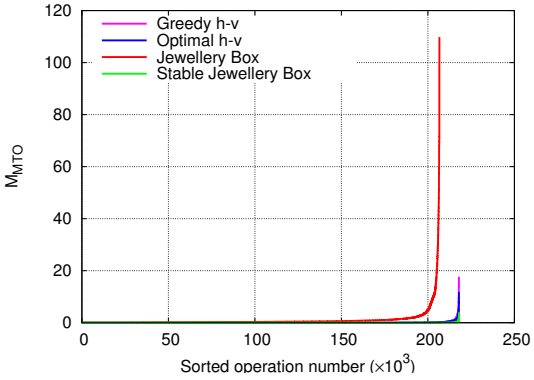


Figure 5.8: Inclusion layout algorithm comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

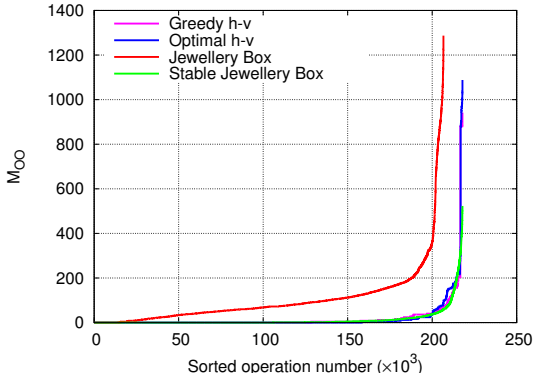


Figure 5.9: Inclusion layout algorithm comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

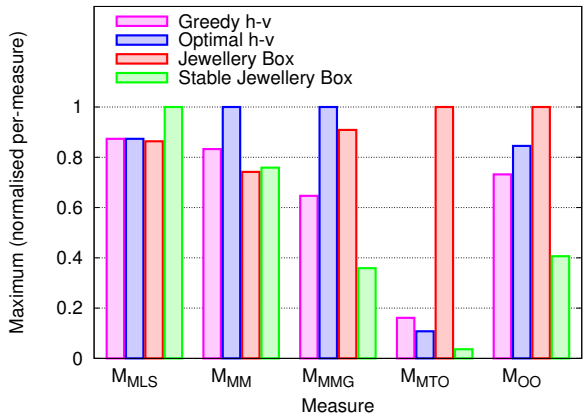


Figure 5.10: Inclusion layout algorithm comparison by using the normalised maximum of each measure.

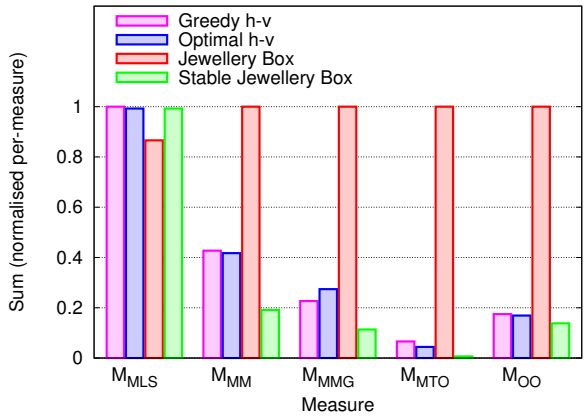


Figure 5.11: Inclusion layout algorithm comparison by using the normalised sum of each measure.

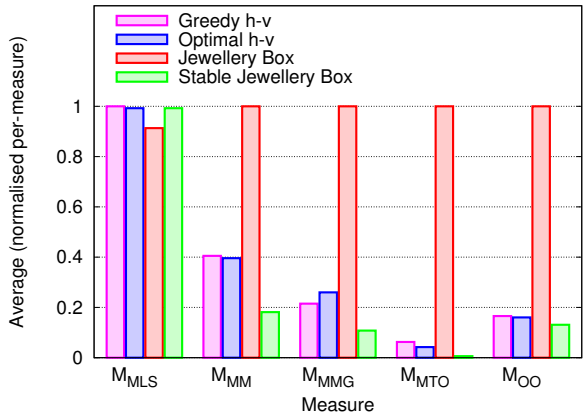


Figure 5.12: Inclusion layout algorithm comparison by using the normalised average of each measure.

5.3.2 Layout algorithm — Node-link style

Figures 5.13 to 5.17 show the sorted results for each quality measure against the choice of layout algorithm in the node-link style. Figures 5.18 to 5.20 show the maximum value, sum and average value summaries of the results.

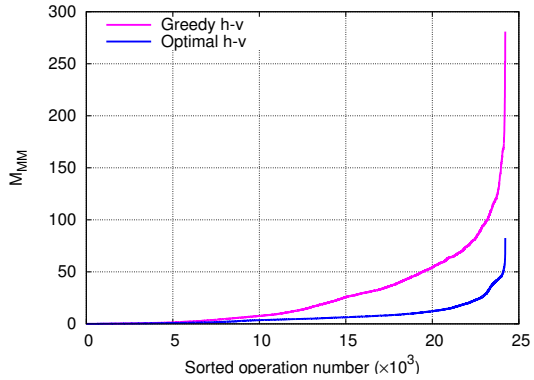


Figure 5.13: Node-link layout algorithm comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

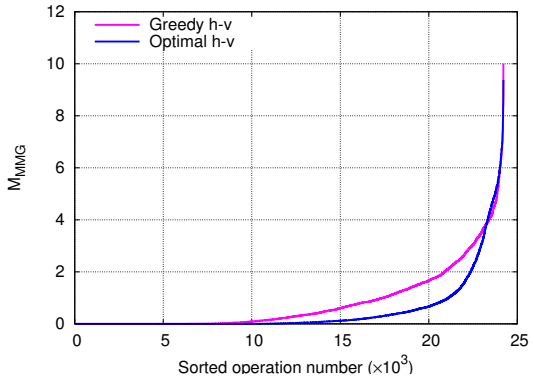


Figure 5.14: Node-link layout algorithm comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

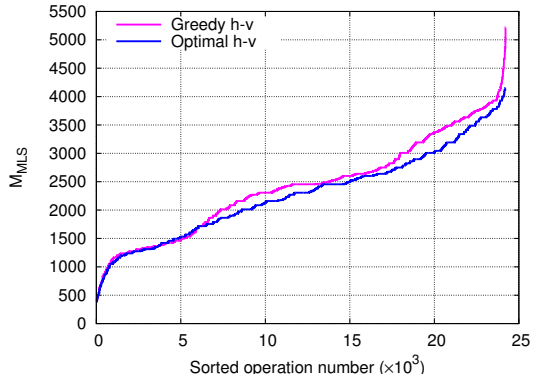


Figure 5.15: Node-link layout algorithm comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

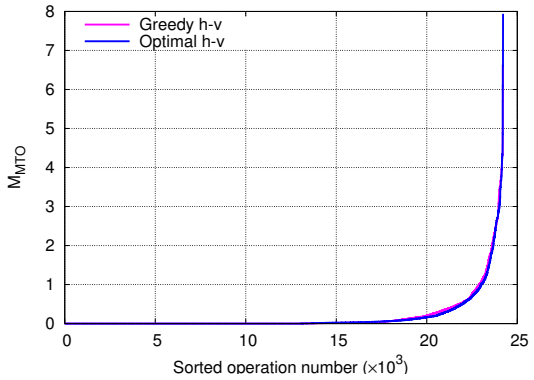


Figure 5.16: Node-link layout algorithm comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

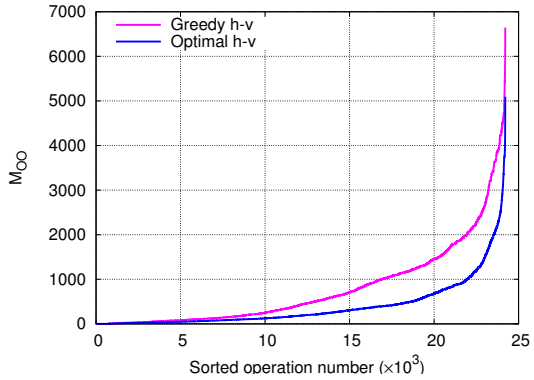


Figure 5.17: Node-link layout algorithm comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

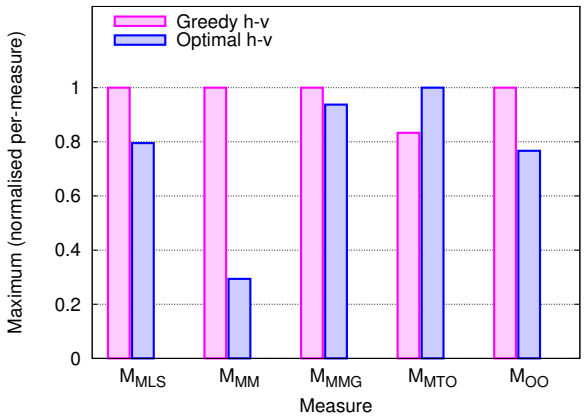


Figure 5.18: Node-link layout algorithm comparison by using the normalised maximum of each measure.

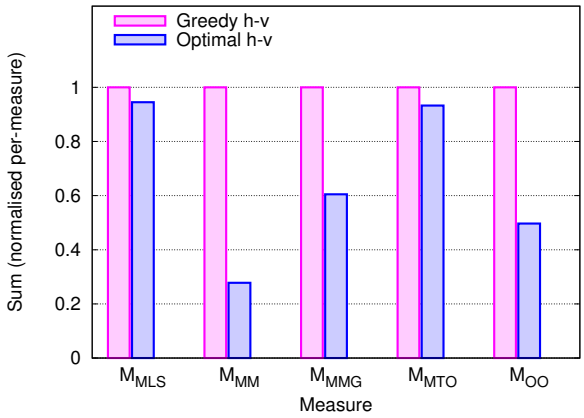


Figure 5.19: Node-link layout algorithm comparison by using the normalised sum of each measure.

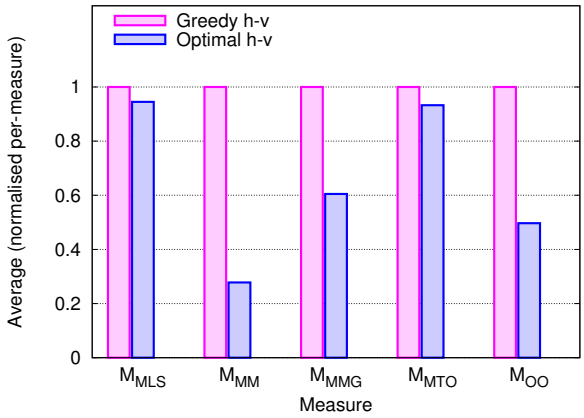


Figure 5.20: Node-link layout algorithm comparison by using the normalised average of each measure.

5.3.3 Maximum detail threshold

Figures 5.21 to 5.25 show the sorted results for each quality measure against the choice of maximum detail threshold. Figures 5.26 to 5.28 show the maximum value, sum and average value summaries of the results.

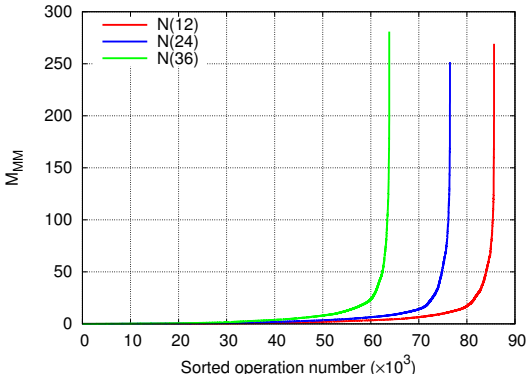


Figure 5.21: Maximum detail threshold comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

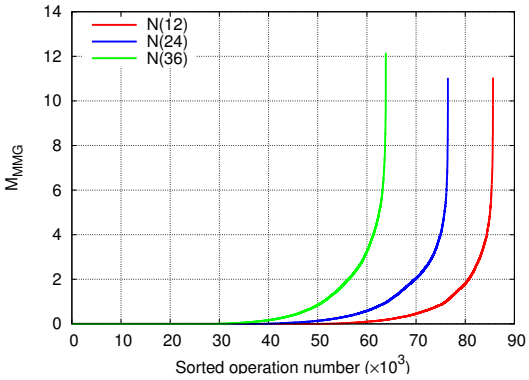


Figure 5.22: Maximum detail threshold comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

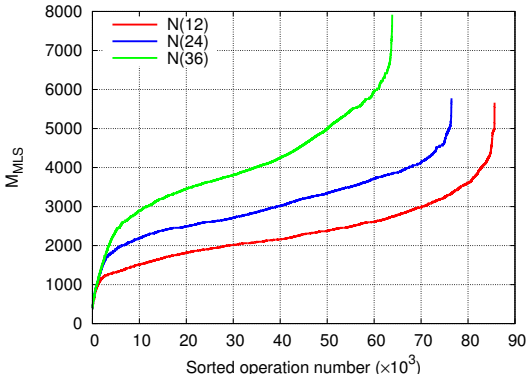


Figure 5.23: Maximum detail threshold comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

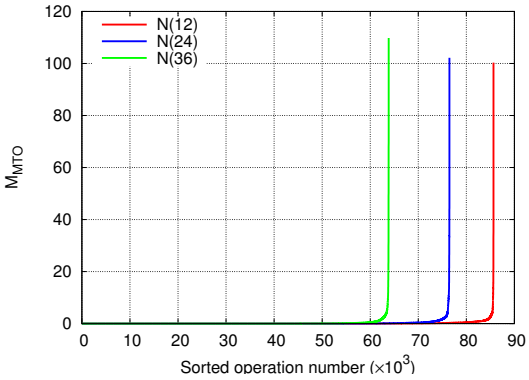


Figure 5.24: Maximum detail threshold comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

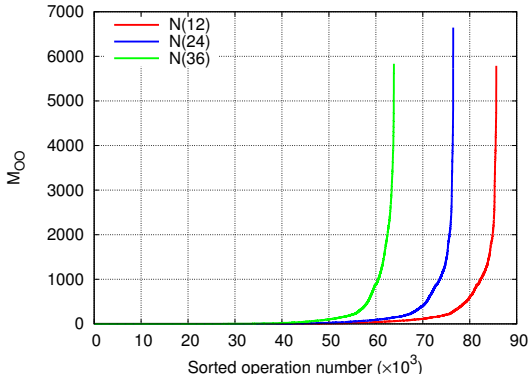


Figure 5.25: Maximum detail threshold comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

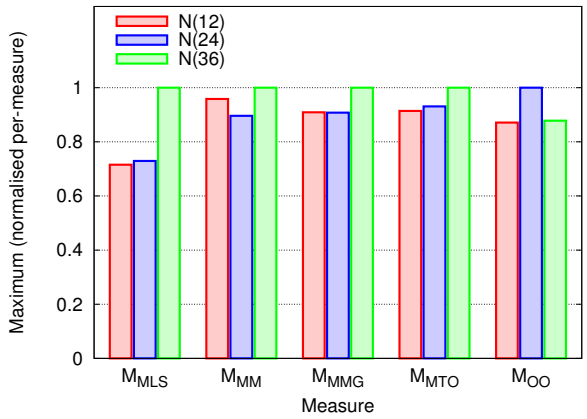


Figure 5.26: Maximum detail threshold comparison by using the normalised maximum of each measure.

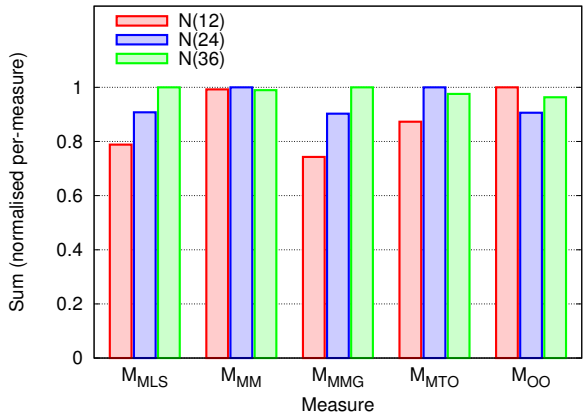


Figure 5.27: Maximum detail threshold comparison by using the normalised sum of each measure.

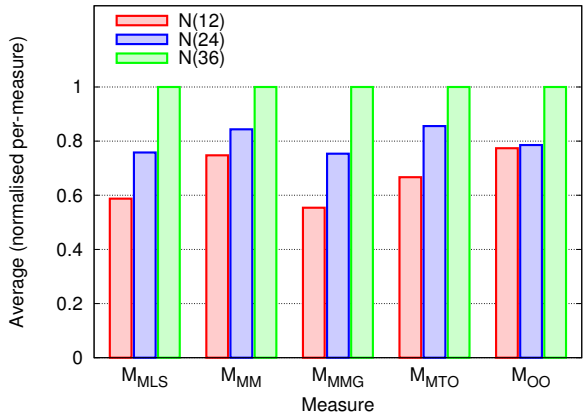


Figure 5.28: Maximum detail threshold comparison by using the normalised average of each measure.

5.3.4 Navigation paths — Target node

Figures 5.29 to 5.33 show the sorted results for each quality measure against the choice between "target node — perfect" and "target node — normal" navigation strategies. Figures 5.34 to 5.36 show the maximum value, sum and average value summaries of the results.

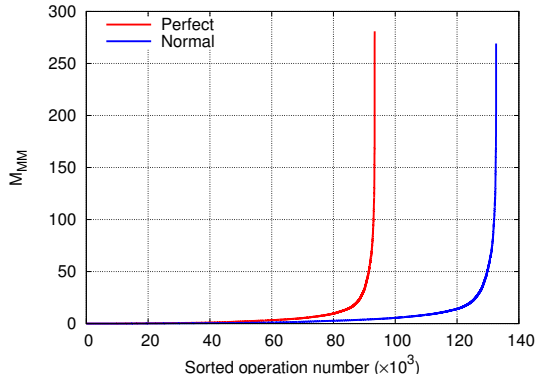


Figure 5.29: Target node navigation strategies comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

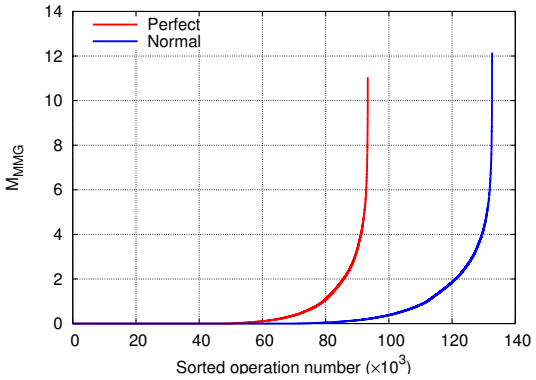


Figure 5.30: Target node navigation strategies comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

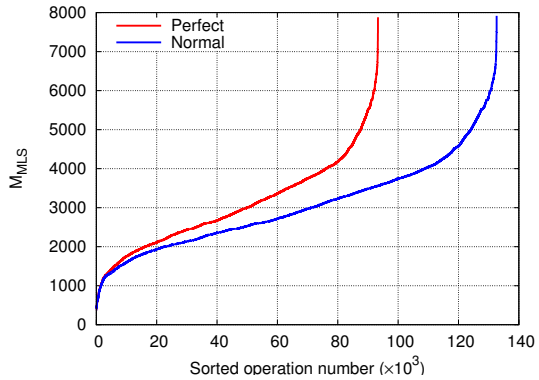


Figure 5.31: Target node navigation strategies comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

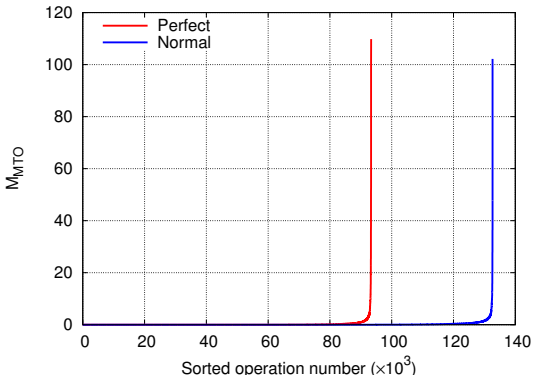


Figure 5.32: Target node navigation strategies comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

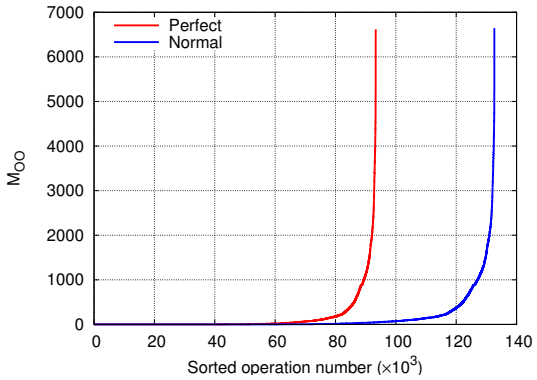


Figure 5.33: Target node navigation strategies comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

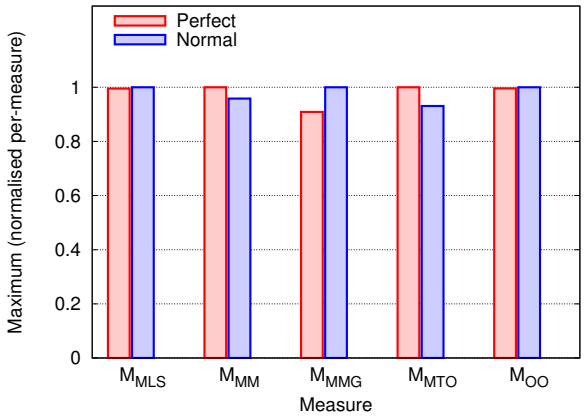


Figure 5.34: Target node navigation strategies comparison by using the normalised maximum of each measure.

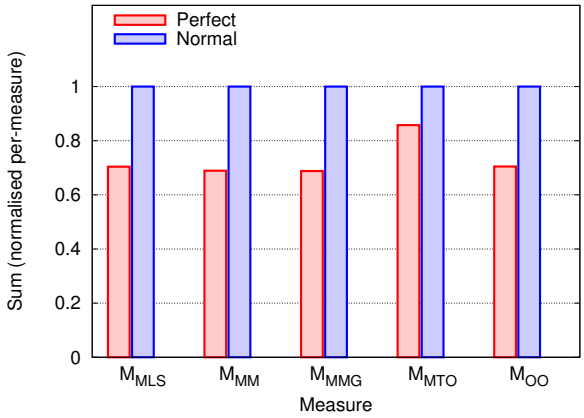


Figure 5.35: Target node navigation strategies comparison by using the normalised sum of each measure.

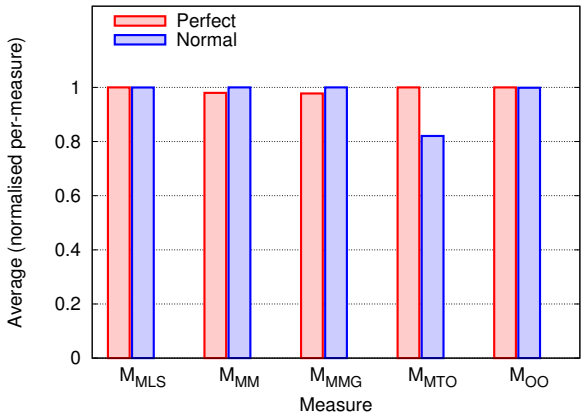


Figure 5.36: Target node navigation strategies comparison by using the normalised average of each measure.

5.3.5 Navigation paths — Sample size

Figures 5.37 to 5.41 show the sorted results for each quality measure against the choice between a random sample size of 25% and 75% for the “target node” navigation strategies. Figures 5.42 to 5.44 show the maximum value, sum and average value summaries of the results.

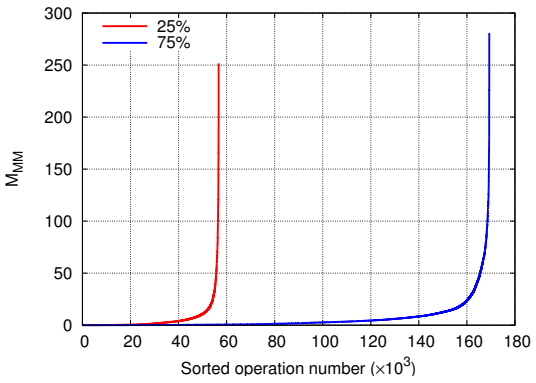


Figure 5.37: Sample size comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

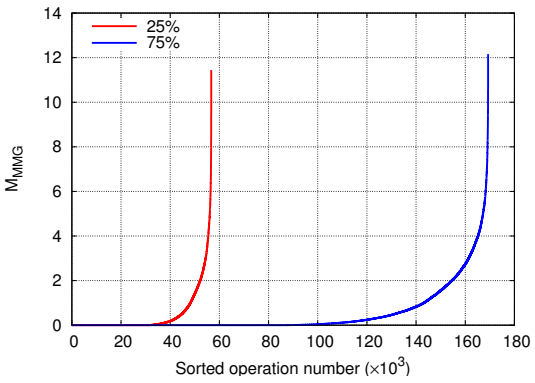


Figure 5.38: Sample size comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

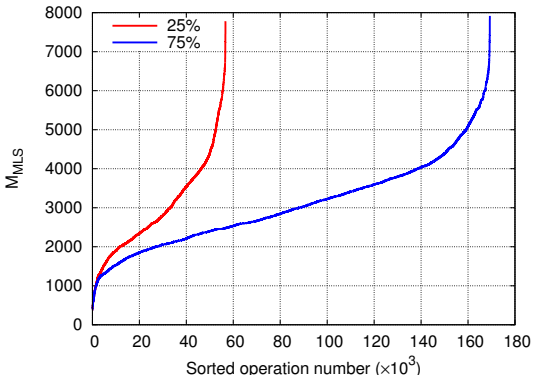


Figure 5.39: Sample size comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

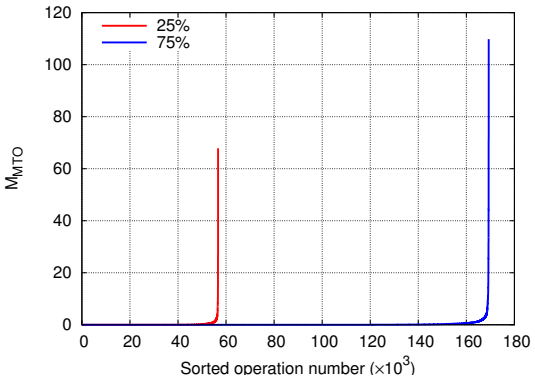


Figure 5.40: Sample size comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

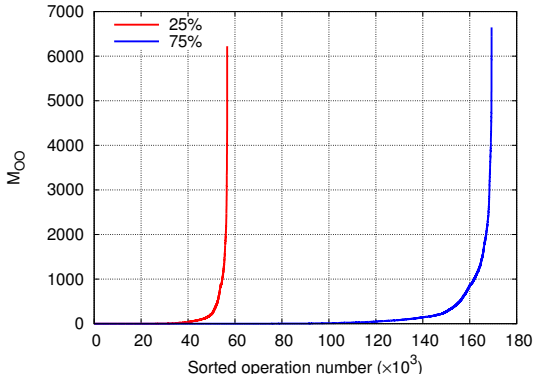


Figure 5.41: Sample size comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

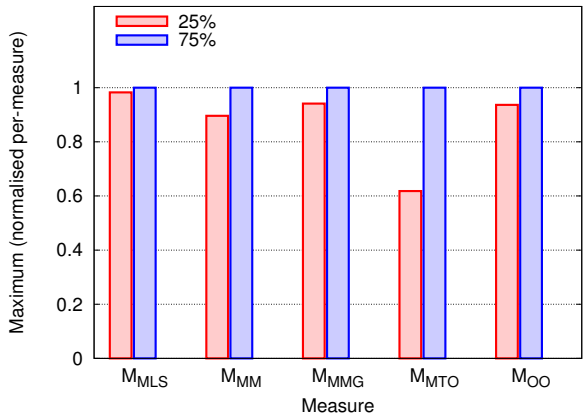


Figure 5.42: Sample size comparison by using the normalised maximum of each measure.

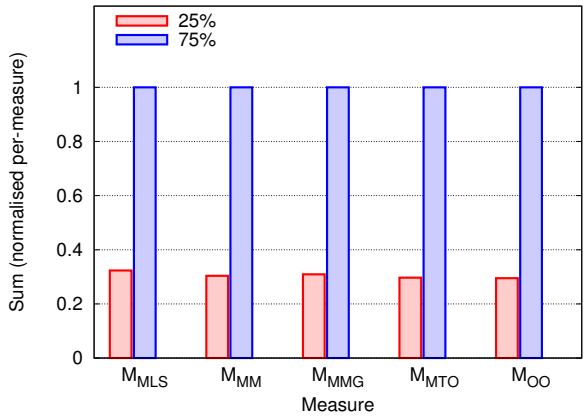


Figure 5.43: Sample size comparison by using the normalised sum of each measure.

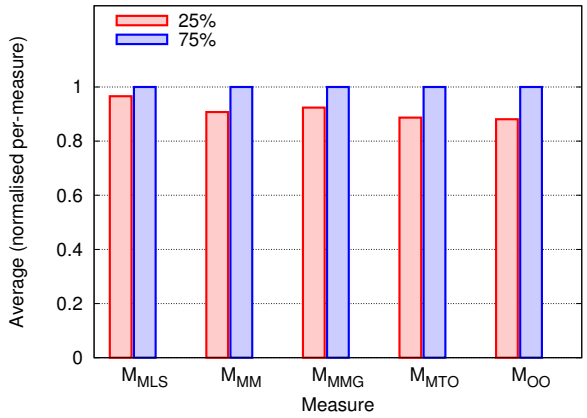


Figure 5.44: Sample size comparison by using the normalised average of each measure.

5.3.6 Node rotation strategy

Figures 5.45 to 5.49 show the sorted results for each quality measure against the three options of node rotation strategy. Figures 5.50 to 5.52 show the maximum value, sum and average value summaries of the results. The results show that for the $\mathcal{M}_{\mathrm{MM}}$ and $\mathcal{M}_{\mathrm{MMG}}$ measures, orthogonal rotation is the poorest, followed by circular and linear rotation. The results for the $\mathcal{M}_{\mathrm{MLS}}$ and $\mathcal{M}_{\mathrm{OO}}$ measures show no difference, and the results for the $\mathcal{M}_{\mathrm{MTO}}$ measure shows that linear rotation is far worse than circular and orthogonal rotation.

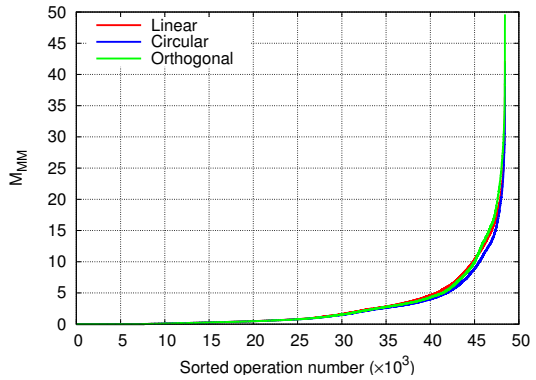


Figure 5.45: Node rotation strategy comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

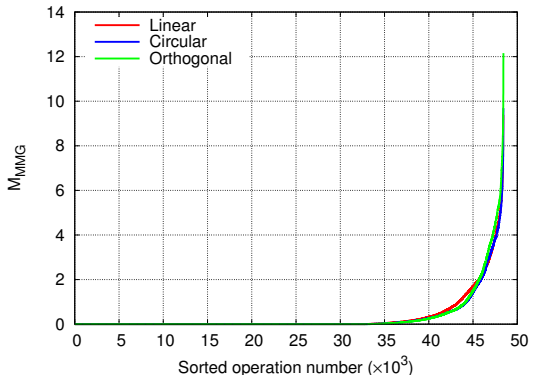


Figure 5.46: Node rotation strategy comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

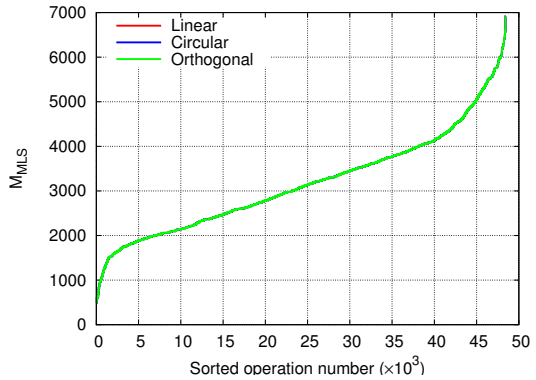


Figure 5.47: Node rotation strategy comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

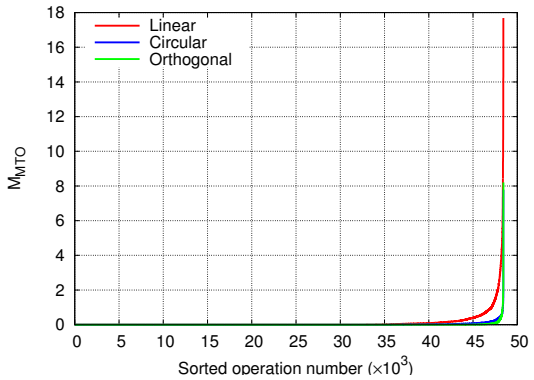


Figure 5.48: Node rotation strategy comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

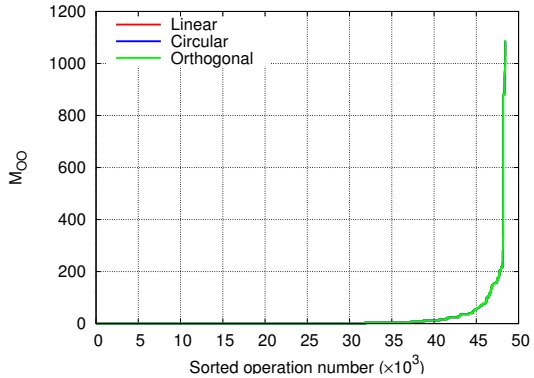


Figure 5.49: Node rotation strategy comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

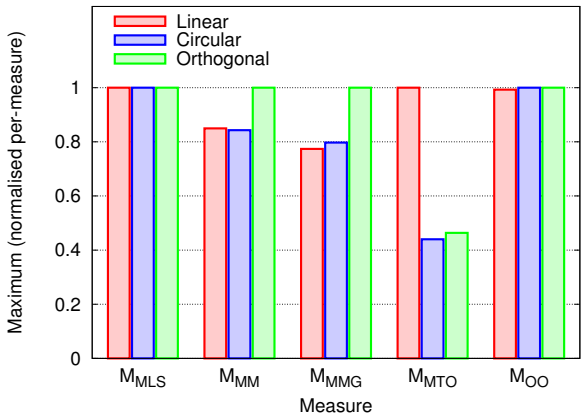


Figure 5.50: Node rotation strategy comparison by using the normalised maximum of each measure.

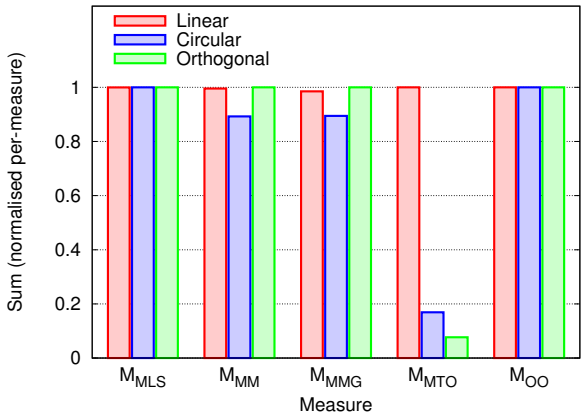


Figure 5.51: Node rotation strategy comparison by using the normalised sum of each measure.

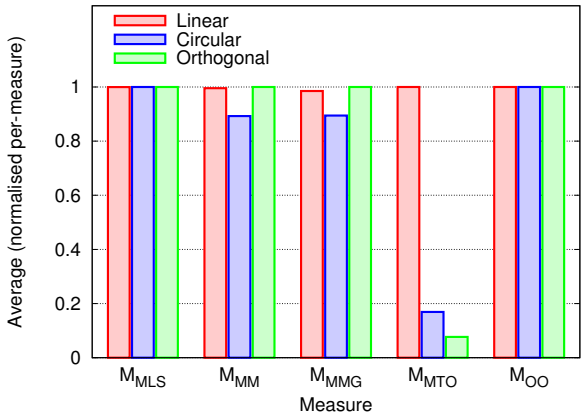


Figure 5.52: Node rotation strategy comparison by using the normalised average of each measure.

CHAPTER 6

Software Views

The second empirical case study of Structural Zooming is that of *software views*. Section 6.1 introduces and describes software views as they are used in this investigation. Section 6.2 describes the data source for the investigation, how the data is represented within the Structural Zooming system, and sets out the experimental design of the empirical investigation. Finally, Section 6.3 presents the results. The conclusions derived from this investigation are analysed and interpreted in Chapter 8, along with those presented in Chapters 5 and 7.

6.1 Background

Chapter 5 presents Design Behaviour Trees: a type of tree from the field of software engineering that represents the behavioural requirements for a proposed piece of software. By contrast, *software views* provide visualisations of pre-existing software systems. The motivation for these visualisations is to aid the software developer to become familiarised with the software quickly. This is particularly useful for tasks such as reverse-engineering, refactoring, the maintenance and re-design of legacy systems, and the training of programmers who have recently begun development on an existing software system.

Software views are created by analysing the structure of the software, followed by the visual presentation of this structure. The analysis can be either "static" or "dynamic" [155]. *Static analysis* determines the software structure by examining the source code or compiled object code, and is also known as *compile-time analysis*. *Dynamic analysis* determines the software structure by examining the actual execution of the software system, and is also known as *run-time analysis*. Both methods have advantages and disadvantages. For example, static analysis requires a parser for each language desired, whereas dynamic analysis is (generally) language-neutral. However, static analysis is independant of the hardware architecture, and in fact does not require any such architecture to exist,

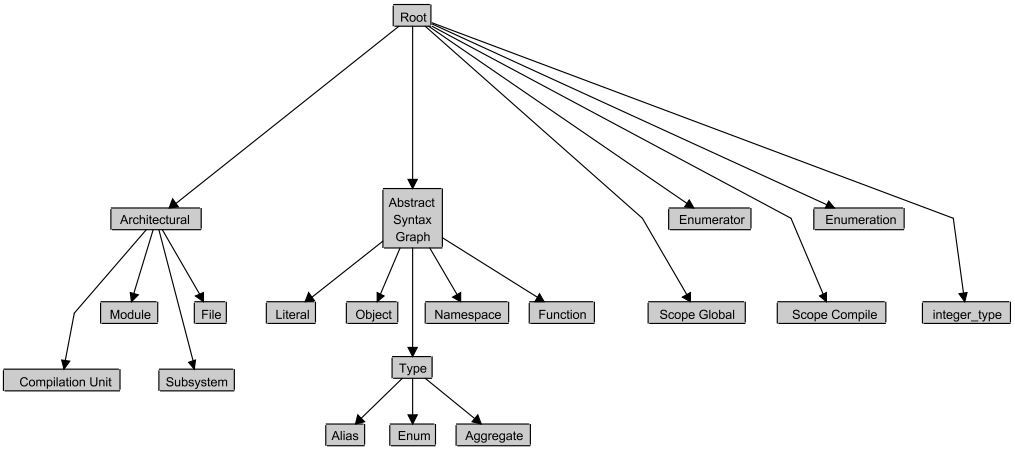


Figure 6.1: The hierarchy of twenty node types used by Swagkit [144].

which is not the case for dynamic analysis. Dynamic analysis is able to give additional information about the performance of the software, indicating which areas are executed more often than others. However, ensuring that all parts of the code are executed during the analysis is a difficult problem.

Once the structure of the software has been analysed, it is represented as a graph. The nodes represent parts of the software system, such as modules, files, classes, functions and variables. An edge joins two nodes to represent the presence of a relationship between those two parts of the software system, such as class inheritance, aggregation of variables, and the caller-callee relationship between functions. A particular *view* of the software is a collection of nodes and edges of certain types. For example, the *call-graph* is a common view which shows the calling relationship among functions, and is thus useful for identifying coupling and dead code.

This chapter examines the application of Structural Zooming to software views.

6.2 Experimental design

In this investigation, a static view of several pieces of freely available utility software is considered. The views are created by using *Swagkit* [144], a collection of tools for parsing and analysing C/C++ source code [24, 31, 48, 73]. It is similar to the widely used *Rigi* [102, 103, 150] toolkit for reverse engineering. Swagkit is a series of patches for the GNU C/C++ compiler (gcc) version 3.0 [147], that expose the internal data structures of the compiler. Supporting scripts allow a C or C++ software package to be compiled as usual, with a view of the entire software package as an additional output.

Swagkit defines twenty different types of nodes, arranged hierarchically as shown in Figure 6.1.

REFERSTO		Meaning
FUNCTION	FUNCTION	Function calls
FUNCTION	OBJECT	Variable references
OBJECT	FUNCTION	Function pointers
OBJECT	OBJECT	Aggregations (`struct`/`union`) and pointers

Table 6.1: Meanings of the different combinations of REFERSTO edges.

However, this investigation makes use of only seven of these node types.

- SUBSYSTEM: A large section of the software, such as that identified by a library file or executable binary.
- MODULE: A collection of code which forms a cohesive unit within the software package.
- FILE: A single file of code in the software package.
- OBJECT: A section of memory allocated to an identifier, such as a variable, array, struct or object.
- FUNCTION: A callable function or subroutine.
- ENUMERATOR: An element in an enumerated type.
- LITERAL: The actual value of an element in an enumerated type.

There are three types of edges.

- CONTAINS: Indicates that the target node is contained within the source node. For example, FUNCTIONs are contained in FILEs, which are contained in MODULEs. Figure 6.2 shows how the types of nodes may be contained within one another.
- REFERSTO: Indicates that the source node refers in some way to the target node. The source and target nodes may be either FUNCTION or OBJECT nodes. Table 6.1 shows the meanings of each of these combinations.
- LINKS: Indicates a link between an identifier declared as `extern` and the location of its actual declaration (that is, the location the identifier will reference after the linking stage of compilation).

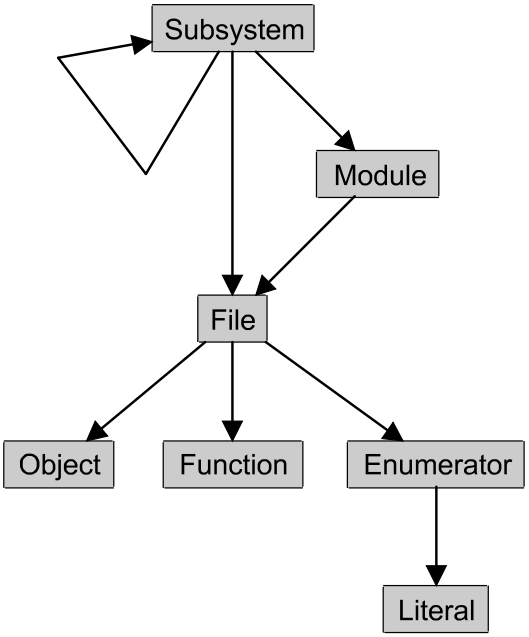


Figure 6.2: The allowed CONTAINS edge types in Swagkit.

The CONTAINS edge is ideally suited for use as the cluster hierarchy in a clustered graph. The REFERSTO and LINKS edges comprise the remainder of the edges in the clustered graph.

Parameters

This investigation uses a corpus of eleven open-source software packages, primarily from the GNU free software project [51]. Each software package was processed using Swagkit to obtain the corpus of data files.

- `bash-2.05b-libglob`: The GNU *Bourne Again SHell*, or *bash*, version 2.05b [52], is a re-implementation of the original UNIX Bourne shell with additional features. It is the standard shell for the GNU project and is the most commonly used shell in Linux distributions. It is well structured into numerous self-contained libraries. This is the "*glob*" library, which resolves filename wildcards.
- `bash-2.05b-libhistory`: As above, except that this is the "*history*" bash library, which maintains a log of user commands and allows previous commands to be recalled, edited and re-executed.

- `bash-2.05b-libmalloc`: As above, except that this is the "*malloc*" bash library, which performs memory management functions for bash.

- `bash-2.05b-libsh`: As above, except that this is the "*sh*" bash library, which implements numerous simple and commonly required functions for shell-style operation, such as string processing and interfacing with the operating system.

- `bash-2.05b-libtilde`: As above, except that this is the "*tilde*" bash library, which provides functions for resolving filesystem directories using the tilde notation for naming home directories.

- `bc-1.06-dc`: The GNU *bc* package, version 1.06 [106], is an arbitrary precision numeric processing language. The *dc* program [117] is a reverse-polish notation calculator that shares much of the bc codebase for its computations.

- `grep-2.5`: The GNU *grep* package, version 2.5 [53], is the GNU project's implementation of the standard UNIX grep utility for efficiently searching text files with regular expressions.

- `hello-1.3`: The GNU *hello* package [50], version 1.3, is an example of how a canonical software package in the GNU project ought to appear. Depite its "hello world" namesake, it actually shows a diverse range of programming language features, in an effort to show programming style by example.

- `hello-2.1.1`: As above, except that this is version 2.1.1 of the *hello* package, which has been updated to reflect language and tool changes since version 1. It dates from 2002, compared to version 1.3 which dates from 1993, 9 years earlier.

- `c488`: The C488 compiler [145] is one of two example software packages offered with Swagkit. It implements a compiler for a simple language similar to Pascal, and has been used for an introductory course in compilers and interpreters at the University of Toronto.

- `small`: This is the second example software package offered with Swagkit. It is an extremely small and trivial piece of software, intended only as a simple example of how Swagkit operates.

Figure 6.3 shows an example screenshot of the `c488` data file using the Stable Jewellery Box inclusion layout algorithm with "soft" edge routing, defined below. Table 6.2 lists some properties of the corpus files.

One navigation strategy is used for each of the files in the corpus. This is the "Target node - perfect" navigation strategy. For each file, the set of leaf node parents is constructed. A random sample of these nodes are selected as the target nodes to be used by the navigation strategy. The navigation strategy runs until the target node is expanded, at which point it advances to the next target node. The state of the Structural Zooming system is not reset between target nodes. The size of the random sample is either 25% of the set of leaf node parents or 20, whichever is smaller. (This is to obtain a reasonably sized sample, while avoiding unnecessarily long runtimes.) Each file uses the same random number generator seed value, to ensure that random choices made by the navigation strategy are consistent for each run of that data file.

Three inclusion tree layout algorithms are used.

- The optimal h-v inclusion layout algorithm, presented in Section 2.2.1, with the orthogonal h-v node rotation strategy, presented in Section 4.1.2.
- The jewellery box inclusion layout algorithm, presented in Section 2.2.2.
- The stable jewellery box inclusion layout algorithm, presented in Section 4.2.1.

As described in Section 2.2.3, the Tom Sawyer Visualization Software edge routing algorithm has parameters for controlling the edge routing algorithm. Two choices are considered for the parameters controlling node positions and sizes.

- *Hard edge routing* is orthogonal edge routing where the node positions and sizes are fixed after being determined by the inclusion tree layout algorithm.
- *Soft edge routing* is orthogonal edge routing where node positions may be changed and node sizes may be increased.

Three possible choices are considered for the detail-reducing threshold, as described in Sections 3.3 and 4.1.1.

- $N(24)$ — At most 24 on-screen nodes.
- $N_E(4)$ — At most 4 expanded on-screen nodes.

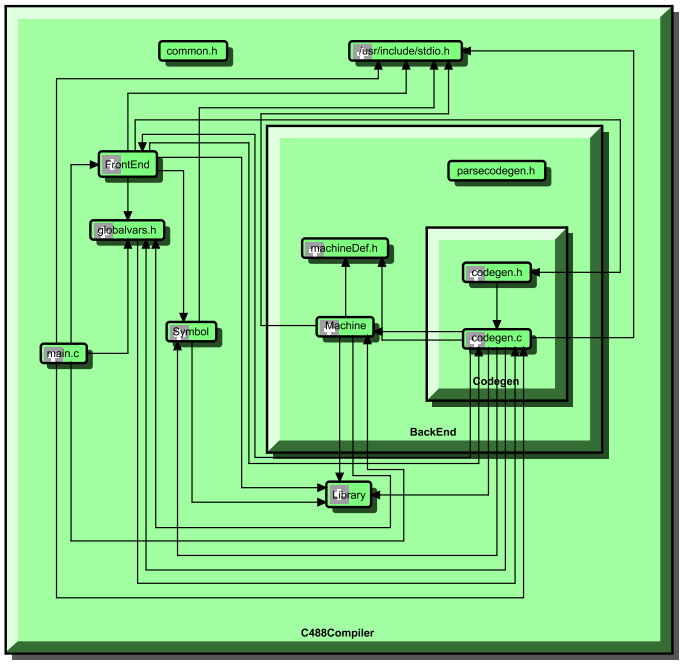


Figure 6.3: An example of the `c488` data file using the Stable Jewellery Box inclusion layout algorithm with soft edge routing.

File	Number of nodes	graph edges	cluster edges	leaf parents	target nodes	lines of code[a]
`bash-2.05b-libglob`	670	147	669	50	13	3349
`bash-2.05b-libhistory`	1025	243	1024	38	10	4955
`bash-2.05b-libmalloc`	627	90	626	83	21	2503
`bash-2.05b-libsh`	1680	280	1679	229	20	8915
`bash-2.05b-libtilde`	415	40	414	11	3	1284
`bc-1.06-dc`	450	340	449	38	10	4305
`grep-2.5`	1072	840	1071	136	20	24054
`hello-1.3`	251	25	250	13	3	1272
`hello-2.1.1`	370	37	369	18	5	7603
`c488`	235	424	234	27	7	3745
`small`	7	2	6	3	1	8

Table 6.2: Properties of the corpus files used in this investigation.

[a] Only lines containing at least three non-whitespace characters are counted.

- $N_E(8)$ — At most 8 expanded on-screen nodes.

As this is an investigation using clustered graphs, the same set of measures as in Chapter 5 are used, with the addition of the edge-based clustered graph measures.

- $\mathcal{M}_{\mathrm{MM}}$ — Minimise Motion
- $\mathcal{M}_{\mathrm{MMG}}$ — Minimise Motion Groups
- $\mathcal{M}_{\mathrm{MLS}}$ — Minimise Layout Size
- $\mathcal{M}_{\mathrm{MTO}}$ — Minimise Transient Occlusions
- $\mathcal{M}_{\mathrm{OO}}$ — Orthogonal Ordering
- $\mathcal{M}_{\mathrm{POB}}$ — Preservation of Bends
- $\mathcal{M}_{\mathrm{POT}}$ — Preservation of Topology

6.3 Results

The results are presented according to each of the individual parameters that were varied, aggregated over the corpus of data files and navigations. Full results are available on the CD accompanying this thesis, as described in Appendix A. A discussion of these results is presented in Chapter 8.

6.3.1 Layout algorithm

Figures 6.4 to 6.10 show the sorted results for each quality measure against the three choices of inclusion layout algorithm. It is clear to see that for all the measures except $\mathcal{M}_{\mathrm{MLS}}$ and $\mathcal{M}_{\mathrm{OO}}$ the Jewellery Box layout algorithm has the worst results, followed by the Stable Jewellery Box algorithm, followed by the optimal h-v layout. In $\mathcal{M}_{\mathrm{MLS}}$ the situation is reversed, where the optimal h-v layout has the worst results, followed by the Stable Jewellery Box algorithm, followed by the Jewellery Box algorithm, while $\mathcal{M}_{\mathrm{OO}}$ is difficult to interpret as no clear trend is discernable.

The corpus-based nature of the investigation naturally suggests using histograms to present the results, aggregating the results and showing their frequency, as opposed to showing a sorted plot of the full results. Figure 6.11 presents the histogram for the $\mathcal{M}_{\mathrm{MM}}$ results from Figure 6.4. It is possible to see in Figure 6.11(a) that the optimal h-v layout has many more instances with low $\mathcal{M}_{\mathrm{MM}}$ value. However, it is hard to compare the Jewellery Box and Stable Jewellery Box algorithms. The

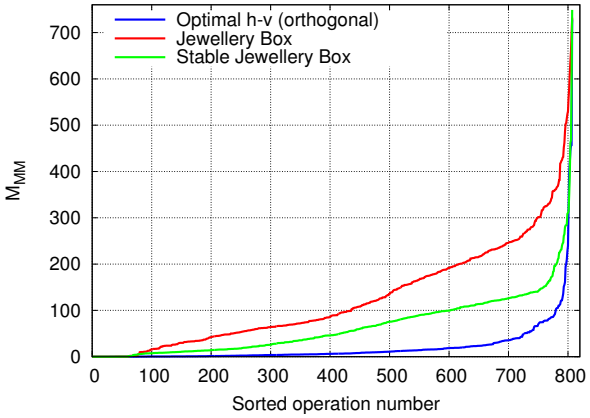


Figure 6.4: Layout algorithm comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

large peak of the optimal h-v layout means that a large range must be accommodated on the scale, making the comparison between smaller values difficult. Figures 6.11(c), 6.11(d) and 6.11(b) show separate bar histograms for each of the layout algorithms. A logarithmic y-axis is used in an attempt to address the large range required. These plots show similar features to Figure 6.11(a), but the logarithmic scale causes unwanted distortion (for example, the Stable Jewellery Box appears to be significantly better than the Jewellery Box for $\mathcal{M}_{\mathrm{MM}}$ values above 150) and comparison is difficult as they are on separate plots. Thus, histograms do not present these results better than sorted line plots.

Figures 6.12 to 6.14 show a good way of summarising the results. Figure 6.12 shows the maximum value for each of the layout algorithms, grouped and independantly normalised by measure. This allows the worst-case performance of the layout algorithms to be directly compared for each measure. Figure 6.13 is similar, except that it shows the sum of each measure rather than the maximum value. This shows the overall performance, and corresponds to the area between the x axis and curve in Figures 6.4 to 6.10. Again, this shows clear relationships between the layout algorithms for each measure. Figure 6.14 shows the average value of each measure, rather than the sum. In this case it shows a similar pattern to the sum in Figure 6.13, however, this view is more useful when the parameters being compared do not have the same number of operations.

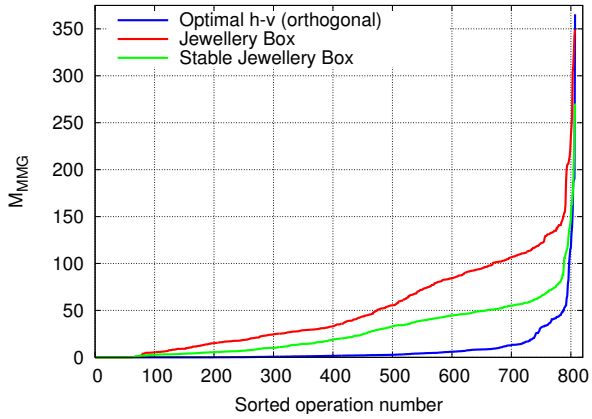


Figure 6.5: Layout algorithm comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

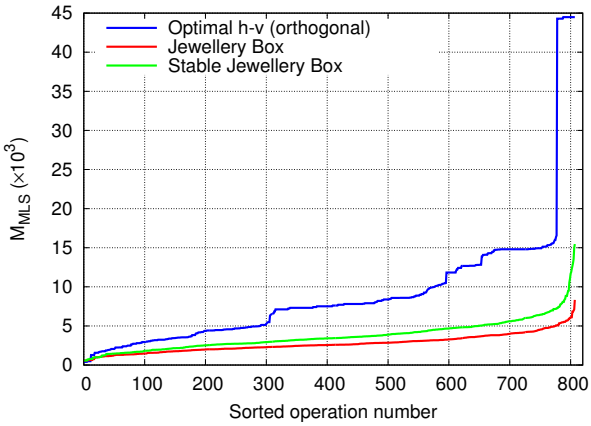


Figure 6.6: Layout algorithm comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

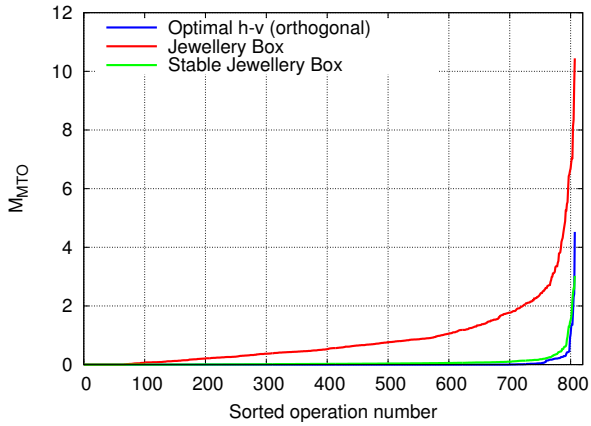


Figure 6.7: Layout algorithm comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

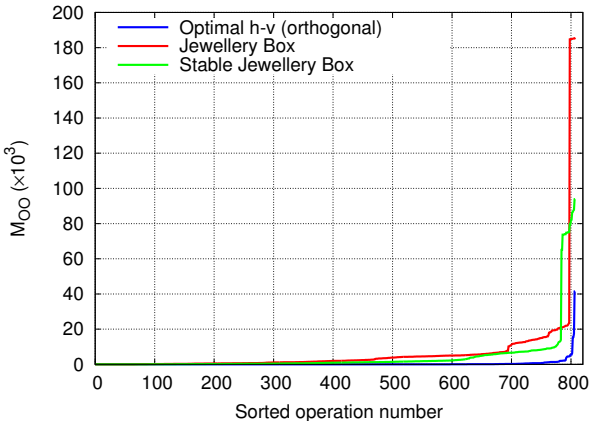


Figure 6.8: Layout algorithm comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

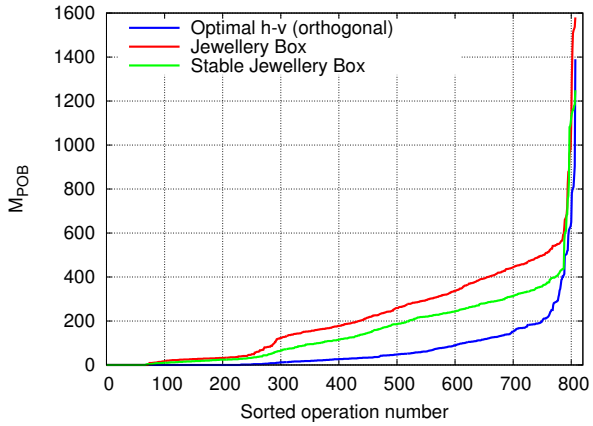


Figure 6.9: Layout algorithm comparison for sorted values of the Preservation of Bends measure $\mathcal{M}_{\mathrm{POB}}$.

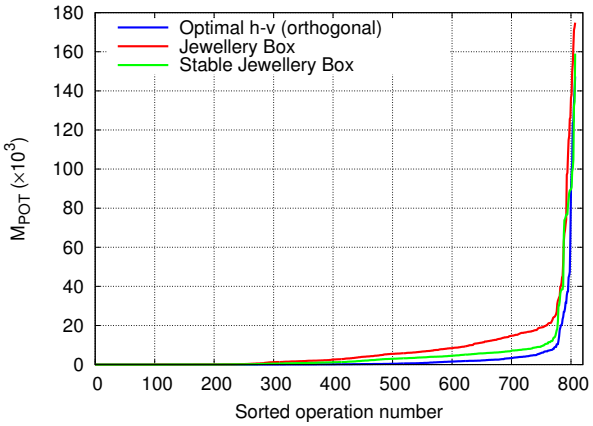
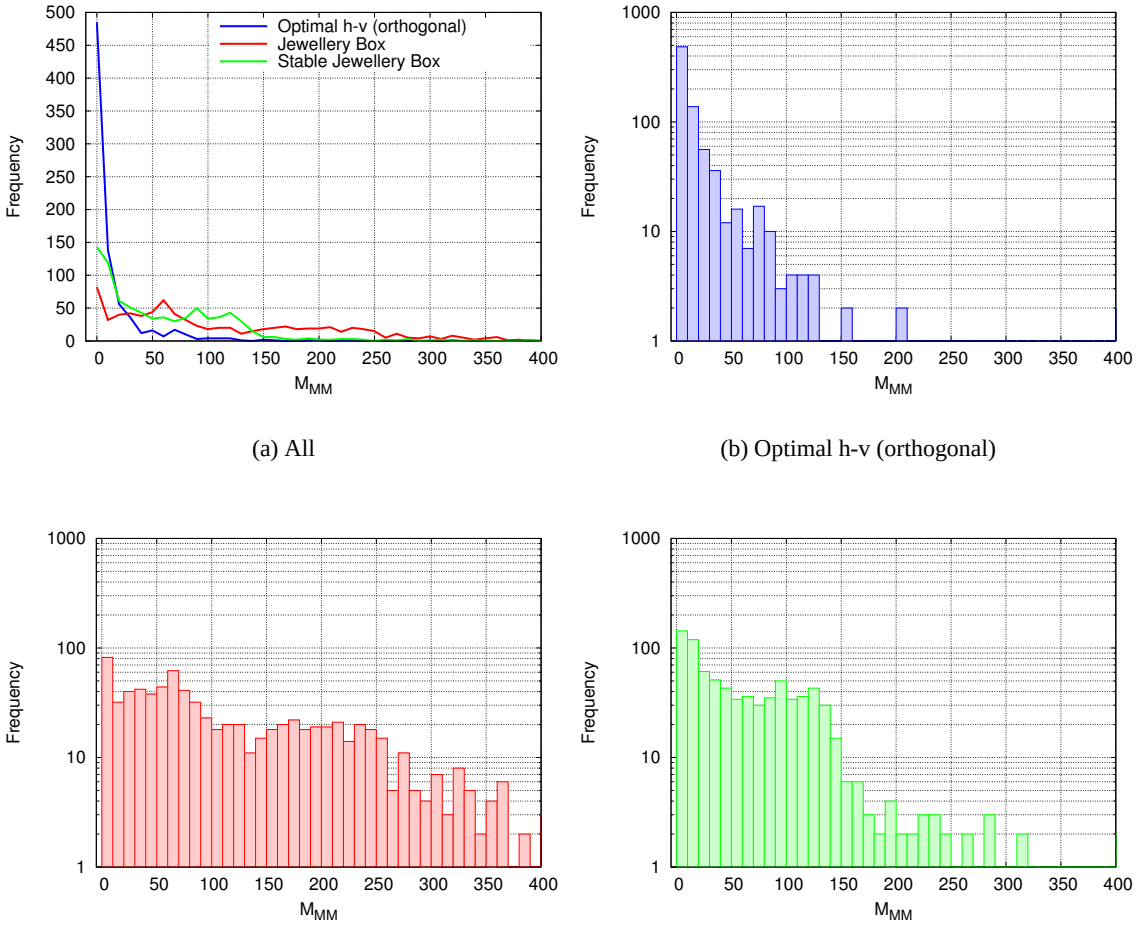


Figure 6.10: Layout algorithm comparison for sorted values of the Preservations of Topology measure $\mathcal{M}_{\mathrm{POT}}$.



(a) All

(b) Optimal h-v (orthogonal)

(c) Jewellery Box

(d) Stable Jewellery Box

Figure 6.11: Layout algorithm comparison using histograms for the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

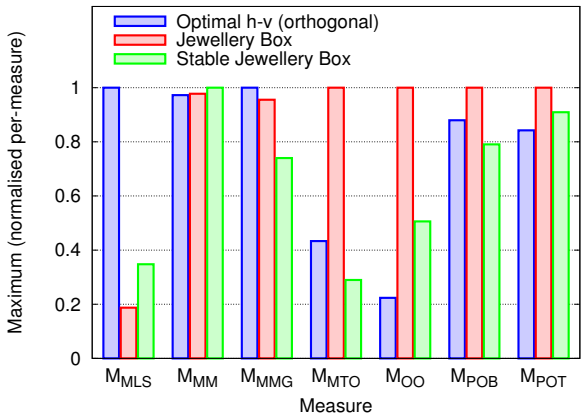


Figure 6.12: Layout algorithm comparison by using the normalised maximum of each measure.

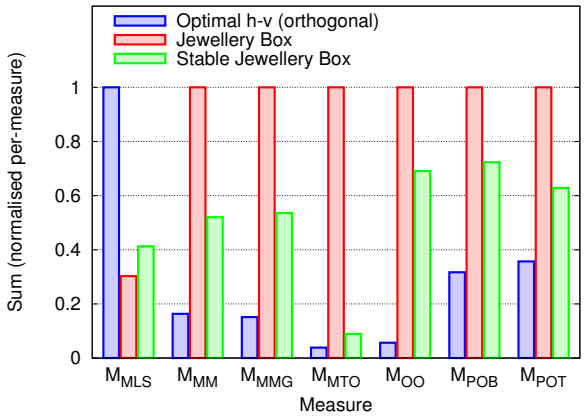


Figure 6.13: Layout algorithm comparison by using the normalised sum of each measure.

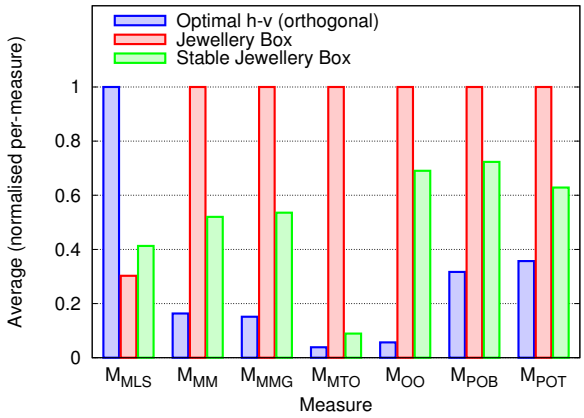


Figure 6.14: Layout algorithm comparison by using the normalised average of each measure.

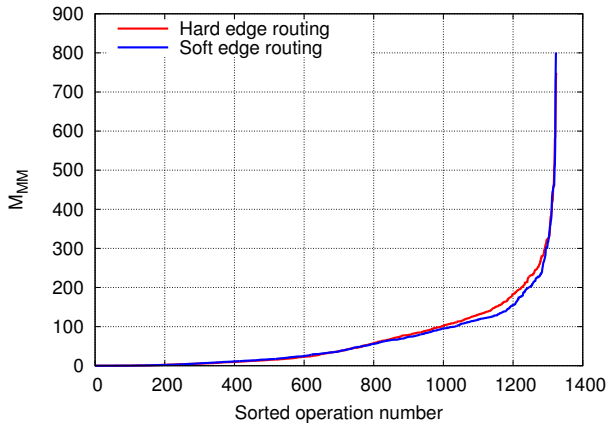


Figure 6.15: Hard vs soft edge routing comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

6.3.2 Hard vs soft edge routing

Figures 6.15 to 6.21 show the sorted results for each quality measure against hard and soft edge routing. Very little difference between hard and soft edge layout is discernable in these results. Figures 6.22 to 6.24 show the summary of these results, which confirms that there is little difference between hard and soft edge routing in any of the measures.

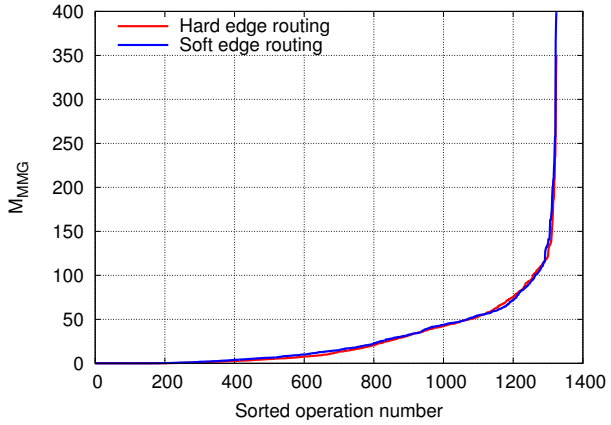


Figure 6.16: Hard vs soft edge routing comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

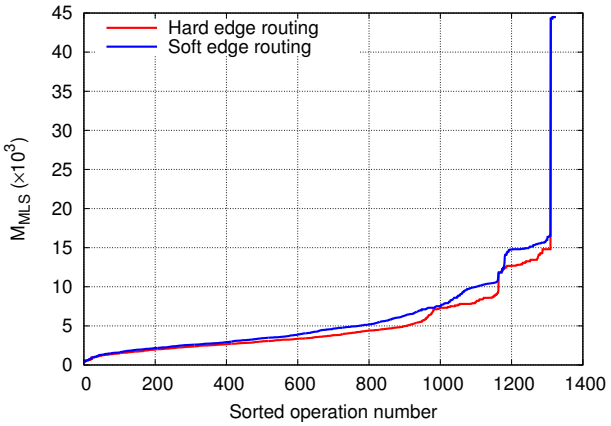


Figure 6.17: Hard vs soft edge routing comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

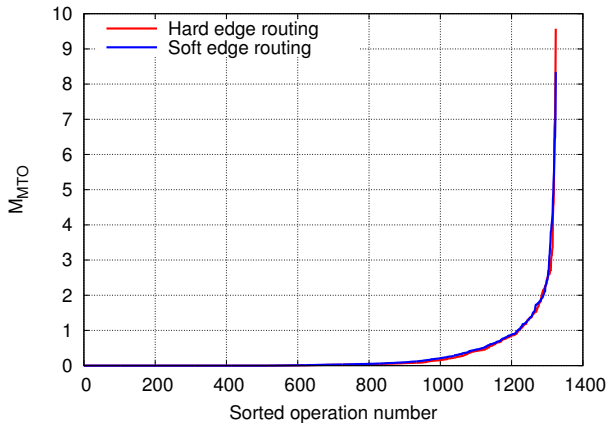


Figure 6.18: Hard vs soft edge routing comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

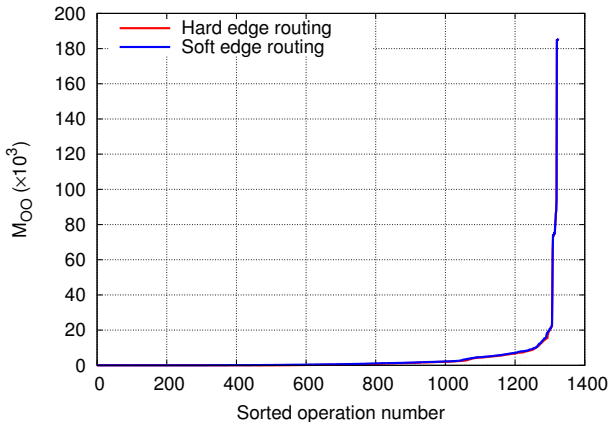


Figure 6.19: Hard vs soft edge routing comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

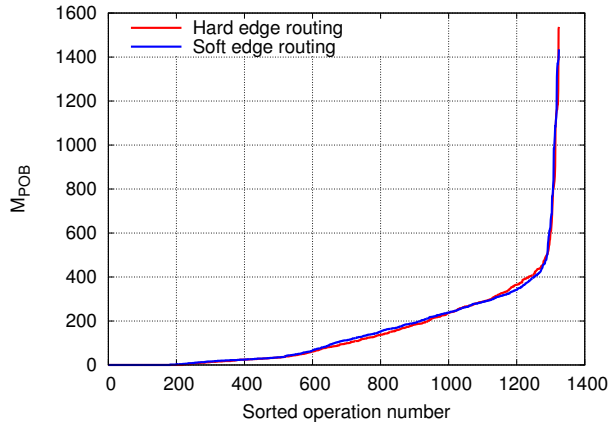


Figure 6.20: Hard vs soft edge routing comparison for sorted values of the Preservation of Bends measure $\mathcal{M}_{\mathrm{POB}}$.

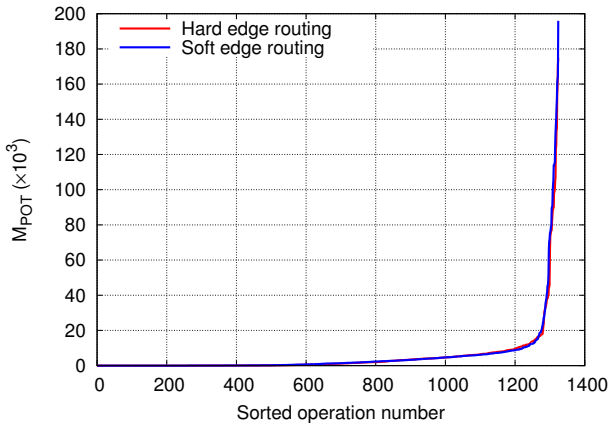


Figure 6.21: Hard vs soft edge routing comparison for sorted values of the Preservation of Topology measure $\mathcal{M}_{\mathrm{POT}}$.

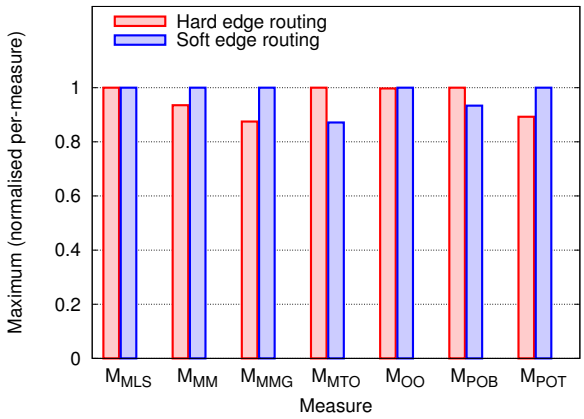


Figure 6.22: Hard vs soft edge routing comparison by using the normalised maximum of each measure.

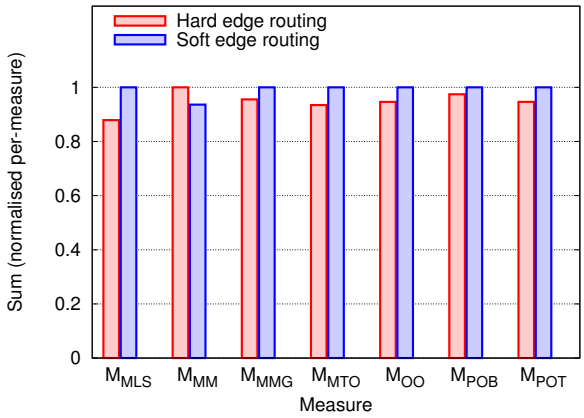


Figure 6.23: Hard vs soft edge routing comparison by using the normalised sum of each measure.

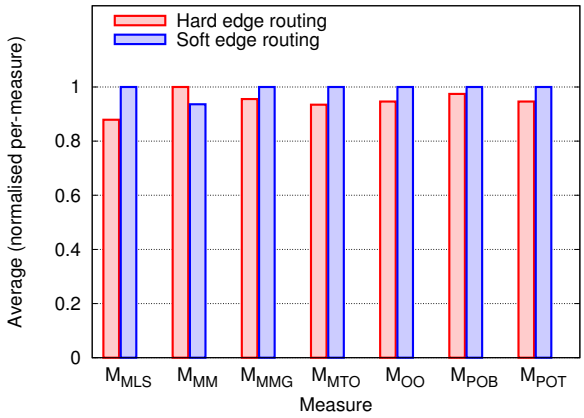


Figure 6.24: Hard vs soft edge routing comparison by using the normalised average of each measure.

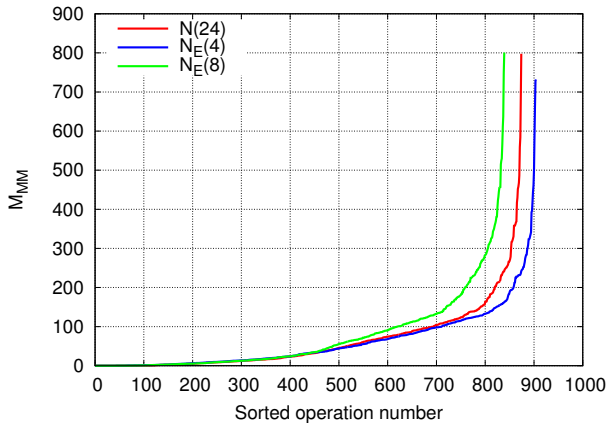


Figure 6.25: Maximum detail threshold comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

6.3.3 Maximum detail threshold

Figures 6.25 to 6.31 show the sorted results for each quality measure against the three choices of maximum detail threshold. Here, the different maximum detail threshold choices have differing numbers of operations. This is because after each target node, the state of the Structural Zooming system may be different for each of the maximum detail threshold choices. This means that some target nodes are reached in fewer operations. For example, if $N_E(8)$ has more on-screen nodes than $N_E(4)$ when a target node is reached then it is likely that the $N_E(8)$ strategy will require fewer steps to reach the next target node. Thus, these results suggest that $N_E(8)$ has the most number of on-screen nodes, followed by $N(24)$, followed by $N_E(4)$. It also seems that the order of decreasing measure values is $N_E(8)$, $N(24)$ and $N_E(4)$, however, the differing numbers of operations make it difficult to determine this, with any certainty, from the sorted plots.

Figures 6.32 to 6.34 show the summary for these results. In particular, Figures 6.33 and 6.34 confirm that the order of decreasing measure values is indeed $N_E(8)$, $N(24)$ and $N_E(4)$, and that $N_E(8)$ is generally noticably worse than $N(24)$ and $N_E(4)$ in all measures.

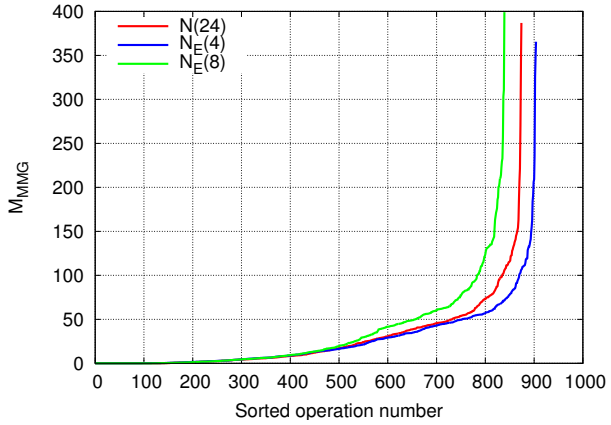


Figure 6.26: Maximum detail threshold comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

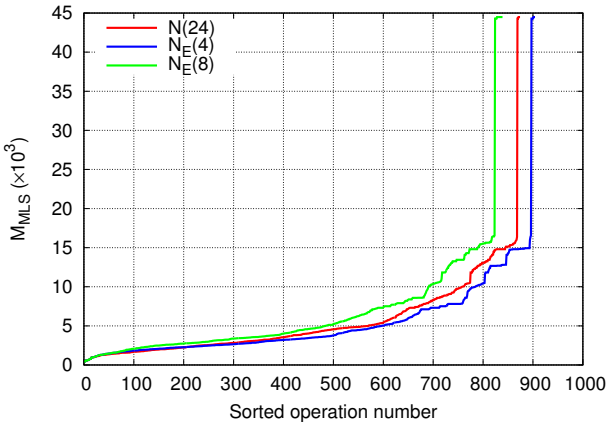


Figure 6.27: Maximum detail threshold comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

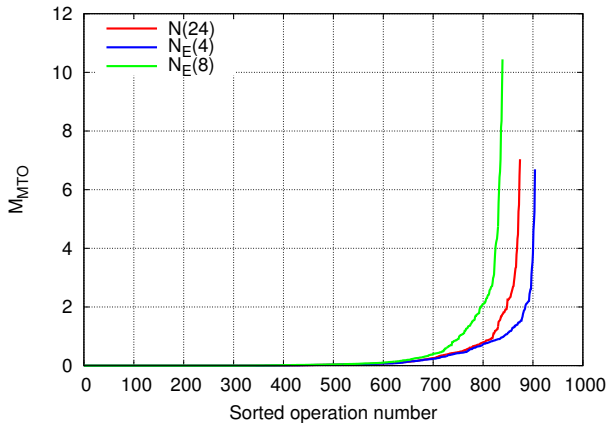


Figure 6.28: Maximum detail threshold comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

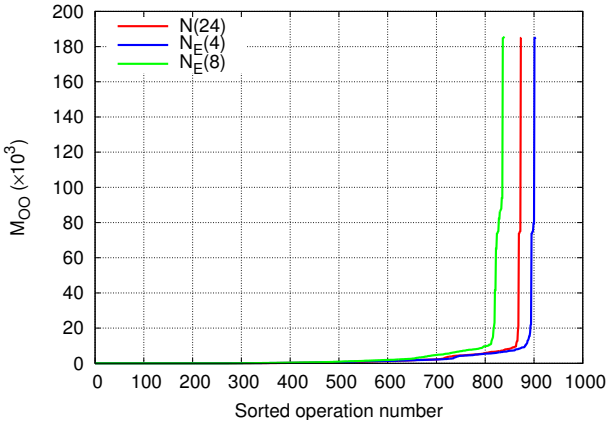


Figure 6.29: Maximum detail threshold comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

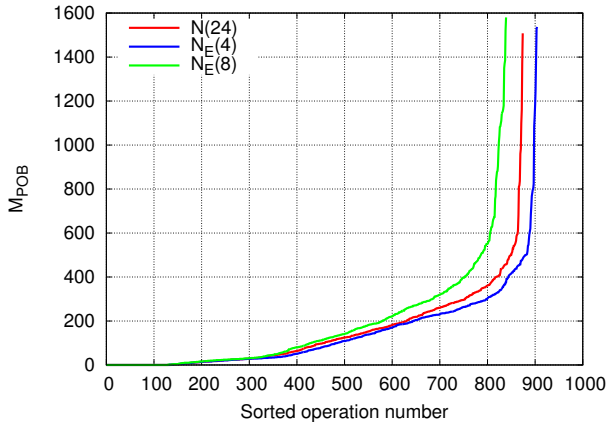


Figure 6.30: Maximum detail threshold comparison for sorted values of the Preservation of Bends measure $\mathcal{M}_{\mathrm{POB}}$.

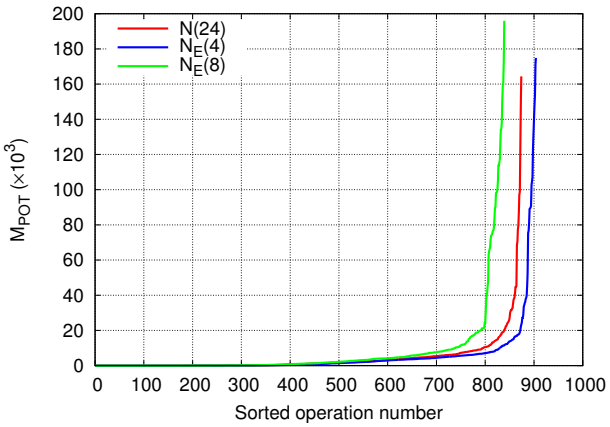


Figure 6.31: Maximum detail threshold comparison for sorted values of the Preservation of Topology measure $\mathcal{M}_{\mathrm{POT}}$.

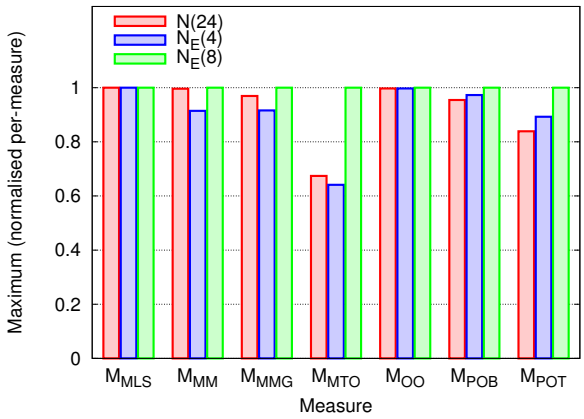


Figure 6.32: Maximum detail threshold comparison by using the normalised maximum of each measure.

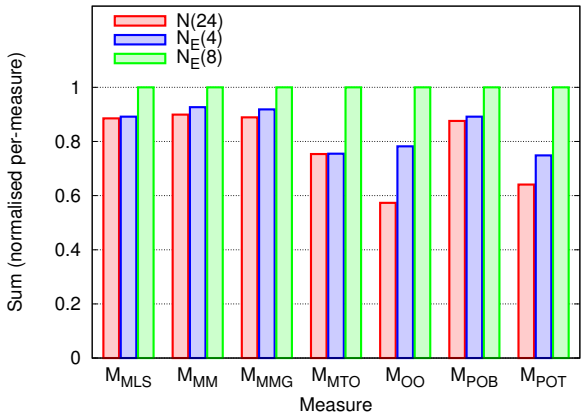


Figure 6.33: Maximum detail threshold comparison by using the normalised sum of each measure.

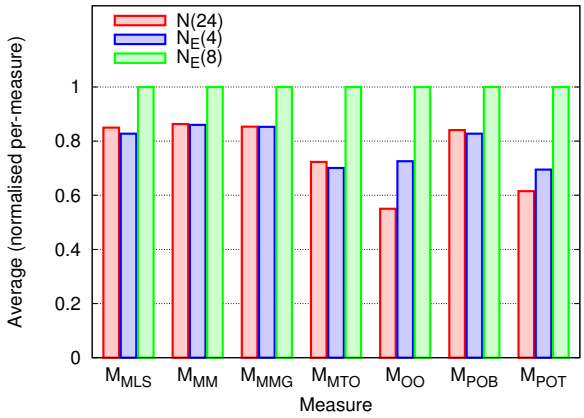


Figure 6.34: Maximum detail threshold comparison by using the normalised average of each measure.

CHAPTER 7

Citation Networks

The third and final empirical case study of Structural Zooming is that of *citation networks*. Section 7.1 introduces and describes citation networks as they are used in this investigation. Section 7.2 describes the data source for the investigation, how the data is represented within the Structural Zooming system, and sets out the experimental design of the empirical investigation. Finally, Section 7.3 presents the results. The conclusions derived from this investigation are analysed and interpreted in Chapter 8, along with those presented in Chapters 5 and 6.

7.1 Background

Social networks is the study of how groups of humans communicate or interact in various situations. The emphasis is on the communication itself — its presence or absence, its size and frequency, any global patterns that can be discerned, and so on. Social networks are typically represented as graphs, where each node represents an *actor*, that may be an individual person or group of people, and an edge between two nodes indicates some form of communication, interaction or relationship between the actors.

For example, the "Kevin Bacon graph" has nodes that represent actors (and actresses), and an edge between two nodes if the actors appeared in a film together. The premise behind the graph is that every actor is on average six edges away from the node representing the actor Kevin Bacon. This is also known as the "six degrees of separation" result, and is studied in the field of *small-world* or *scale-free* networks [9].

Social networks are interesting for several reasons.

Large data source: Humans are social beings and thrive on communication and interaction with one another. This communication takes all manner of forms,

between all manner of people throughout the world. This means that there is always a large amount of social network data available for visualisation.

Rich data source: Social networks are temporal, that is to say, they change over time. In addition, there can be many different types of nodes and edges in any given social network. This richness provides many interesting problems in trying to visualise and understand social networks.

Optimisation in business: The possible communication channels in a team increase with the square of the number of team members. Social networks can help businesses to analyse and optimise their organisational structure from a communication standpoint. Social network analysis also allows businesses to identify particular types of workers, such as domain experts, troubleshooters, people who prefer to work independently, and so on.

Universal relevance: Finally, the concept of social networks is one that has relevance to ordinary people and their lives.

Citation networks are a particular type of social network. A citation network is a graph where each node is a scientific or academic publication, and an edge exists betwee two nodes if one contains a *citation* to the other. Citations are important in scholarly work because they allow scientists and researchers to extend the work of others, allowing a *body of knowledge* to be built. Citation networks allow the analysis of the growth of fields of study, collaborations between researchers in different fields, the identification of important and significant publications, and so on.

The data for this investigation is obtained from the Association for Computing Machinery (ACM) Digital Library [7]. This is a large and mostly complete repository of all ACM publications, including magazines, transactions, journals and conference proceedings. For most publications, the full text of individual articles is available only to ACM Digital Library subscribers. However, bibliographic, abstract and referencing details are publicly available, and this is the source of the citation network data used in this empirical investigation.

7.2 Experimental design

Ordinarily, a citation network is a standard graph. However, many of the documents in the ACM Digital Library have been *classified* by their authors. This classification is in terms of a well-defined

hierarchy of subject matter, designed by the ACM, known as the ACM *Computing Classification System* (*CCS*) [6]. The CCS is used here as the cluster hierarchy of the clustered graph, placing documents as leaf nodes within their appropriate primary classification. Only documents with such a classification have been used. The web page for each document contains a set of "Reference" links to other documents referenced by it. A document may also optionally have a set of "Citation" links to other documents that reference it. Graph edges are created using these reference and citation links, and the direction of the edges is always towards the document being referenced.

Each data file in the corpus is the citation network obtained by selecting an initial document and including the documents which are at most three links from this (ignoring the direction of edges). However, the complete CCS hierarchy consists of over 3200 classes, most of which do not contain any actual document nodes. In order to alleviate this problem, the citation network is "pruned" by removing any class nodes that have no descendant document nodes. In addition, since most documents are classified under the leaf nodes of the CCS, it is often the case that non-leaf cluster nodes have degree of 1. This is handled by "compressing" these non-leaf nodes, as described in Section 2.3.1. Doing so, causes the text classification labels of the compressed nodes to be concatenated, so that the user can still see which classifications are involved. Finally, many of the classification names and document titles can be quite long, creating collapsed and leaf nodes of extreme aspect ratios. To avoid this problem, the text labels of nodes are truncated to at most 40 characters, in two lines of at most 26 characters. Where possible, the line break occurs between words, and truncated text is indicated with an ellipsis at the end of the text.

Parameters

Table 7.1 lists the six corpus data files and some of their properties. Figure 7.1 shows an example screenshot of the `586566` data file using the Stable Jewellery Box inclusion layout algorithm with "soft" edge routing.

	Initial document		Number of						
	ID	cite	doc. nodes	cat. nodes	total nodes	comp. nodes	graph edges	leaf parents	target nodes
a	820113	[143]	16	18	34	15	21	14	4
b	551884	[32]	59	48	107	21	97	36	9
c	774855	[62]	66	60	126	21	110	46	12
d	586566	[91]	131	107	238	40	204	80	20
e	289380	[112]	135	73	208	34	224	59	15
f	564041	[115]	209	123	332	38	386	107	15

Table 7.1: Properties of the corpus files used in this investigation.

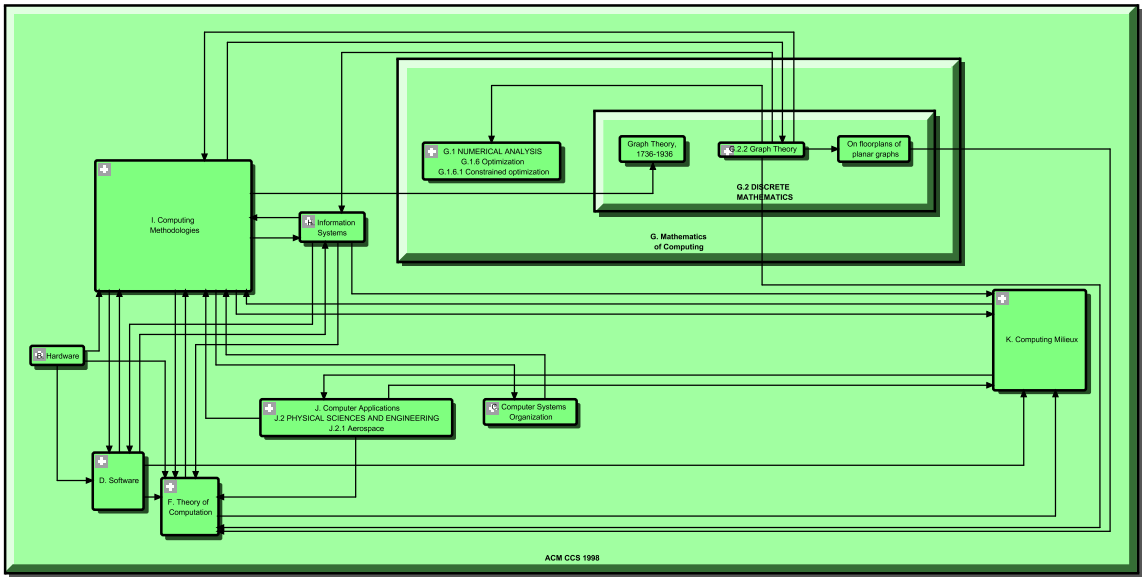


Figure 7.1: An example of the 586566 data file using the Stable Jewellery Box inclusion layout algorithm with soft edge routing.

The parameters used in this case study are similar to those in the study of software views, described in Section 6.2. Specifically,

- The "Target node – perfect" navigation strategy is used.
- A random sample of 25% of the parents of the leaf nodes is used as the target nodes (limited to at most 15 nodes, due to the increased connectivity of the clustered graphs in this corpus).
- Three inclusion layout algorithms are used, being the optimal h-v inclusion layout, the jewellery box inclusion layout algorithm, and the stable jewellery box inclusion layout algorithm.
- "Hard" and "soft" edge routing.
- Three options for the detail-reducing threshold, being $N(24)$ (at most 24 on-screen nodes), $N_E(4)$ (at most four expanded on-screen nodes), and $N_E(8)$ (at most eight expanded on-screen nodes).

The same measures are used, that is,

- $\mathcal{M}_{\mathrm{MM}}$ — Minimise Motion
- $\mathcal{M}_{\mathrm{MMG}}$ — Minimise Motion Groups
- $\mathcal{M}_{\mathrm{MLS}}$ — Minimise Layout Size
- $\mathcal{M}_{\mathrm{MTO}}$ — Minimise Transient Occlusions
- $\mathcal{M}_{\mathrm{OO}}$ — Orthogonal Ordering
- $\mathcal{M}_{\mathrm{POB}}$ — Preservation of Bends
- $\mathcal{M}_{\mathrm{POT}}$ — Preservation of Topology

7.3 Results

As in Section 6.3, the results are presented as a sorted list for each parameter, over the entire corpus. Full results are available on the CD accompanying this thesis, as described in Appendix A. A discussion of these results is presented in Chapter 8.

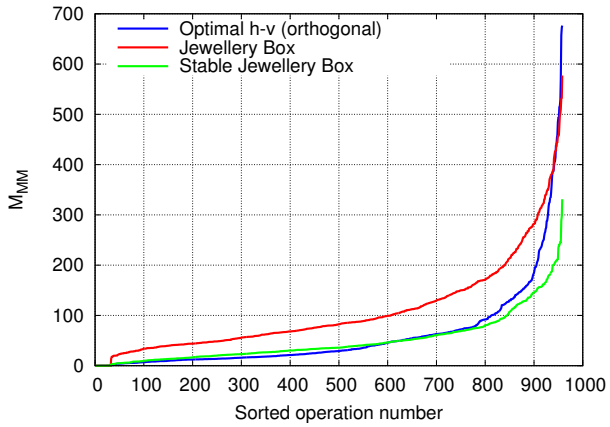


Figure 7.2: Layout algorithm comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

7.3.1 Layout algorithm

Figures 7.2 to 7.8 show the sorted results for each quality measure against the three choices of inclusion layout algorithm. Figures 7.9 to 7.11 summarise these results by showing the maximum value, sum and average value for each measure, as described in Section 6.3.

With the exception of the $\mathcal{M}_{\mathrm{MLS}}$ measure, the worst case of SJBILA is considerably better than the JBILA and the optimal h-v layout. The sum and average values show that the JBILA is worse than the SJBILA and the optimal h-v layout, and the SJBILA is slightly better than the h-v layout. For the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$, the results show that the JBILA gives the smallest layout, followed by the SJBILA and the optimal h-v layout.

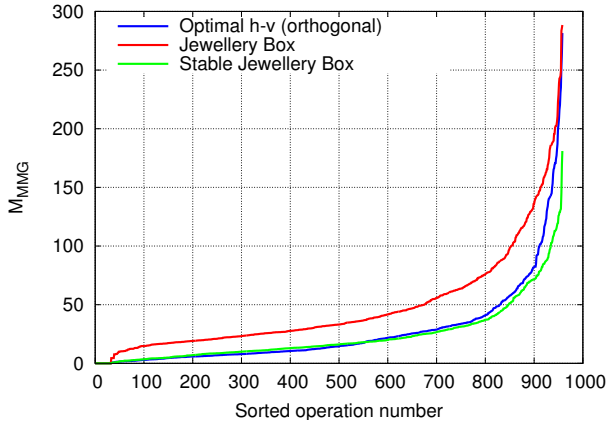


Figure 7.3: Layout algorithm comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

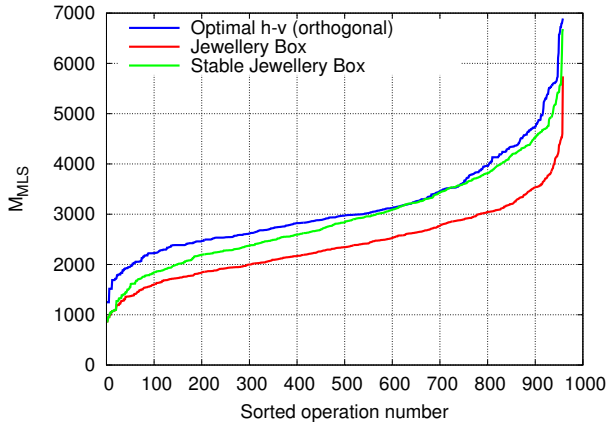


Figure 7.4: Layout algorithm comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

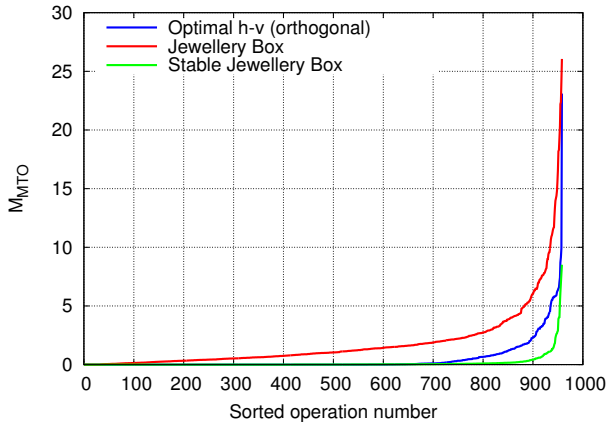


Figure 7.5: Layout algorithm comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

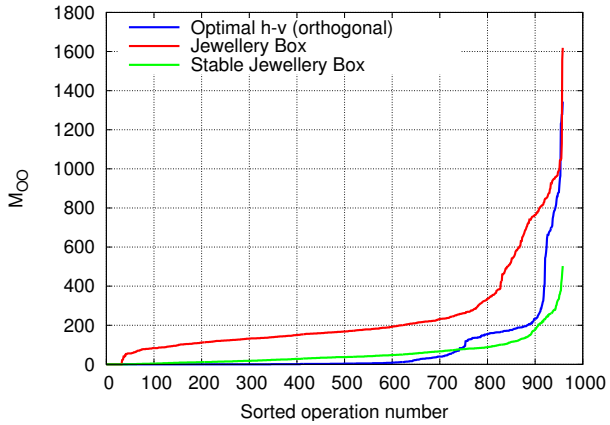


Figure 7.6: Layout algorithm comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

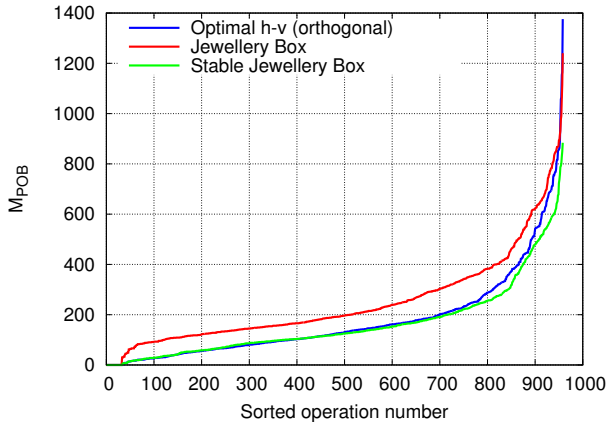


Figure 7.7: Layout algorithm comparison for sorted values of the Preservation of Bends measure $\mathcal{M}_{\mathrm{POB}}$.

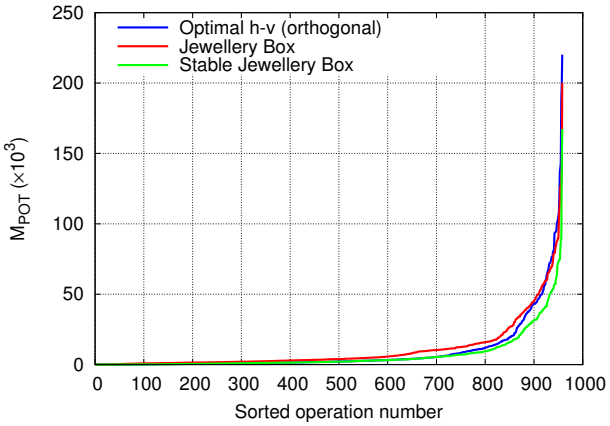


Figure 7.8: Layout algorithm comparison for sorted values of the Preservations of Topology measure $\mathcal{M}_{\mathrm{POT}}$.

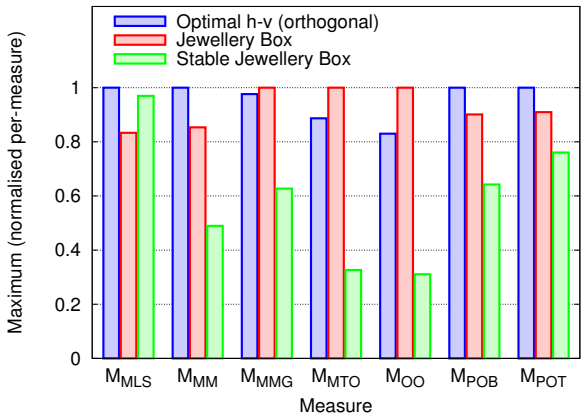


Figure 7.9: Layout algorithm comparison by using the normalised maximum of each measure.

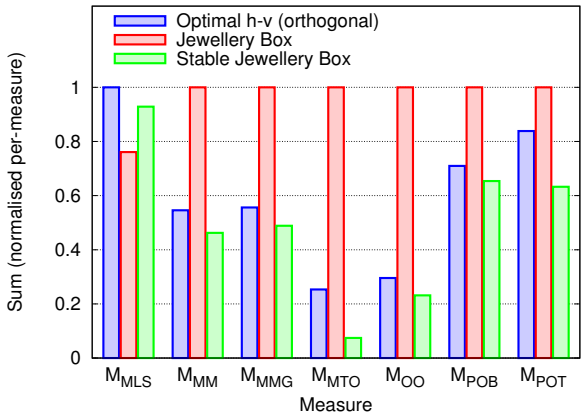


Figure 7.10: Layout algorithm comparison by using the normalised sum of each measure.

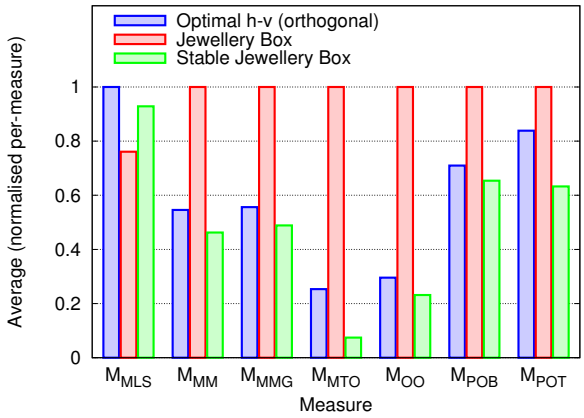


Figure 7.11: Layout algorithm comparison by using the normalised average of each measure.

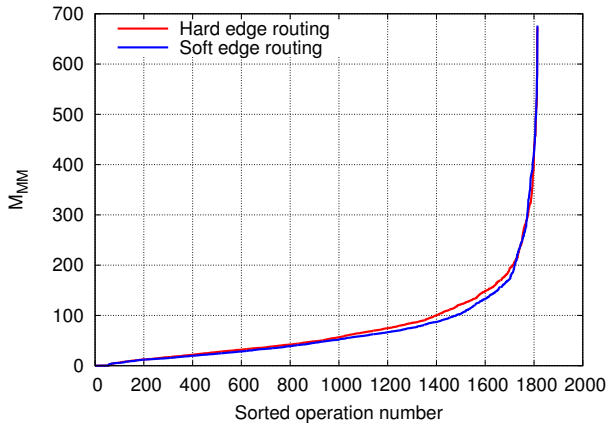


Figure 7.12: Hard vs soft edge routing comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

7.3.2 Hard vs soft edge routing

Figures 7.12 to 7.18 show the sorted results for each quality measure against hard and soft edge routing, and Figures 7.19 to 7.21 show the summary of these results. As in Section 6.3, the difference between hard and soft edge routing is very small. However, it is noticable that in the worst case (that is, the maximum value), hard edge routing is better for the $\mathcal{M}_{\mathrm{OO}}$ and $\mathcal{M}_{\mathrm{POB}}$ measures, and soft edge routing is slightly better for the $\mathcal{M}_{\mathrm{MTO}}$ and $\mathcal{M}_{\mathrm{POT}}$ measures.

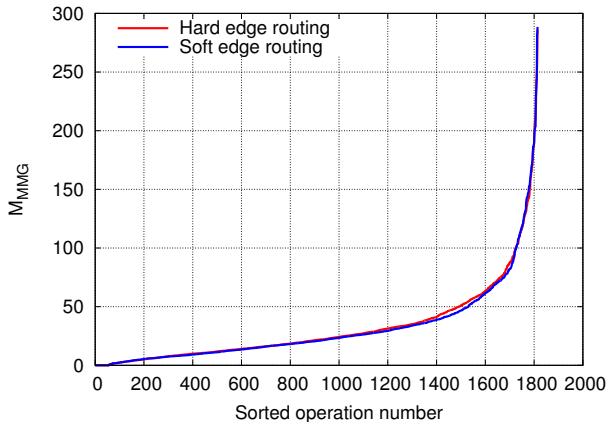


Figure 7.13: Hard vs soft edge routing comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

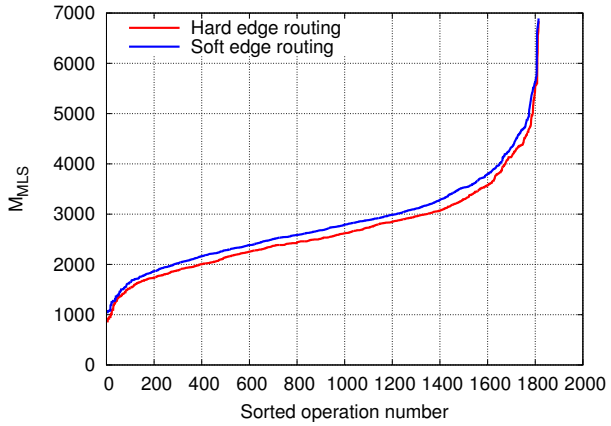


Figure 7.14: Hard vs soft edge routing comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

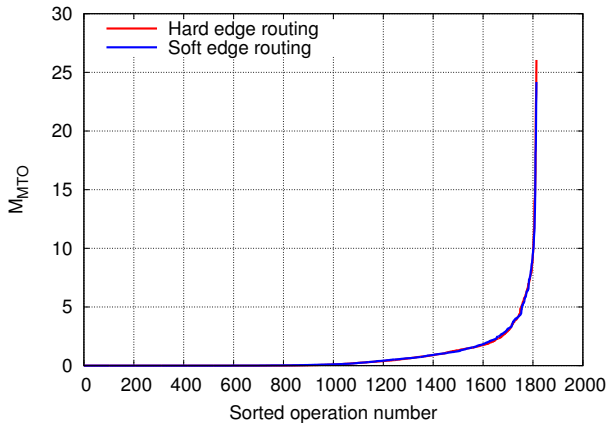


Figure 7.15: Hard vs soft edge routing comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

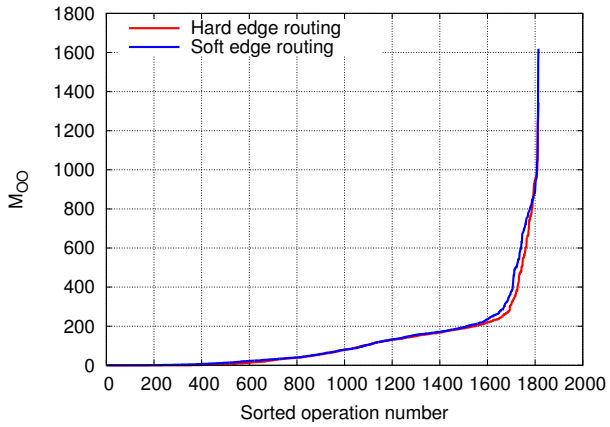


Figure 7.16: Hard vs soft edge routing comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

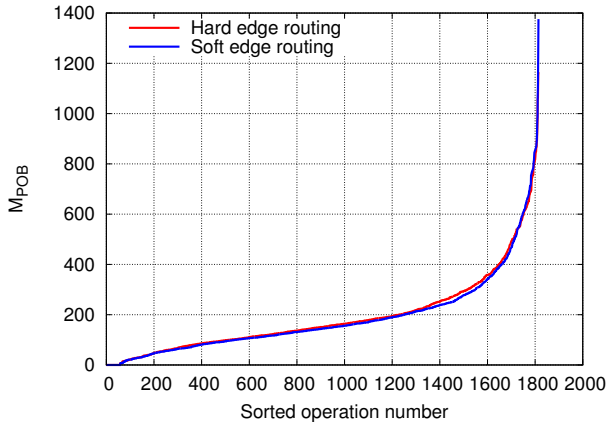


Figure 7.17: Hard vs soft edge routing comparison for sorted values of the Preservation of Bends measure $\mathcal{M}_{\mathrm{POB}}$.

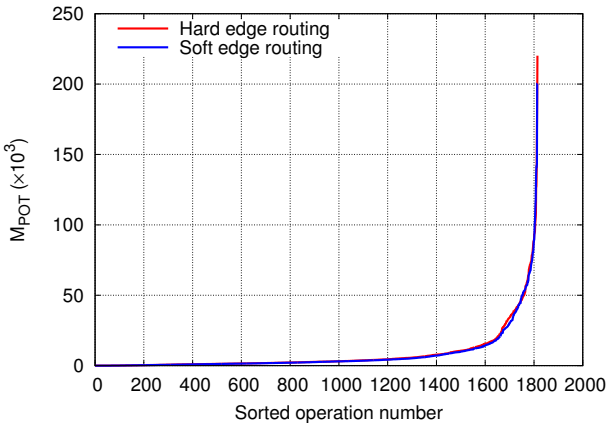


Figure 7.18: Hard vs soft edge routing comparison for sorted values of the Preservation of Topology measure $\mathcal{M}_{\mathrm{POT}}$.

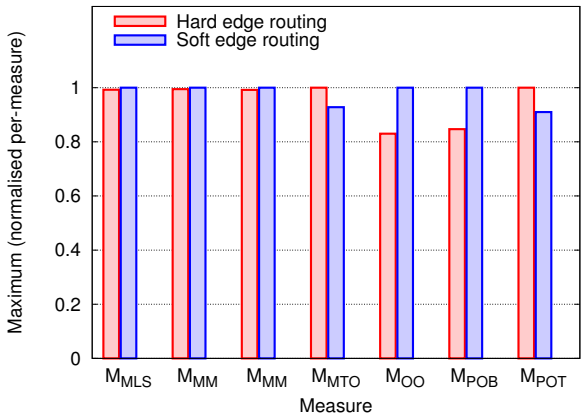


Figure 7.19: Hard vs soft edge routing comparison by using the normalised maximum of each measure.

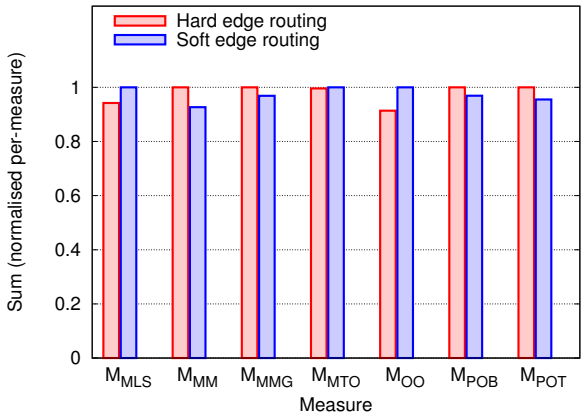


Figure 7.20: Hard vs soft edge routing comparison by using the normalised sum of each measure.

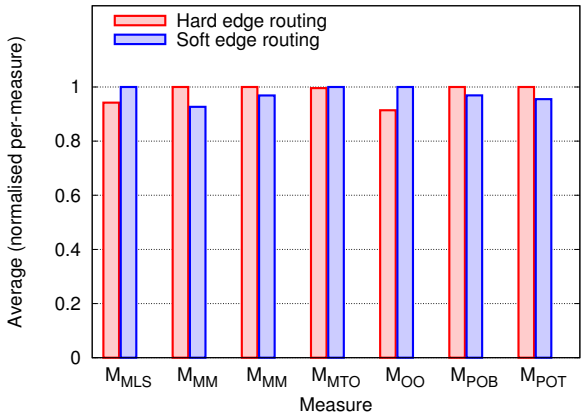


Figure 7.21: Hard vs soft edge routing comparison by using the normalised average of each measure.

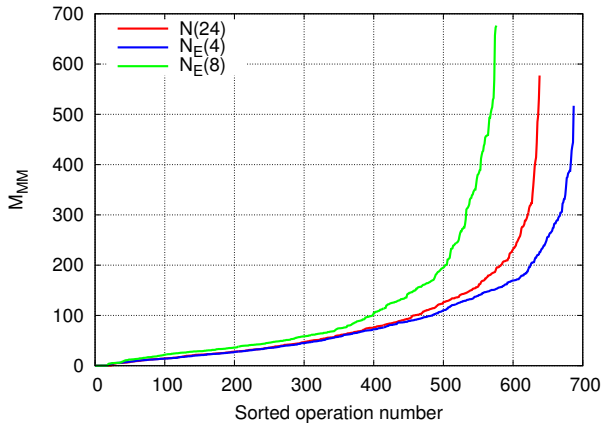


Figure 7.22: Maximum detail threshold comparison for sorted values of the Minimise Motion measure $\mathcal{M}_{\mathrm{MM}}$.

7.3.3 Maximum detail threshold

Figures 7.22 to 7.28 show the sorted results for each quality measure against the three choices of maximum detail threshold, and Figures 7.29 to 7.31 show the summary of these results. It is clear from these results that the worst case of $N_E(8)$ is generally worse than $N(24)$ and $N_E(4)$, particularly for $\mathcal{M}_{\mathrm{MTO}}$, $\mathcal{M}_{\mathrm{OO}}$ and $\mathcal{M}_{\mathrm{POB}}$. The sums of the measures do not vary as much, but $N_E(8)$ is still the worst, or very close to it. The average values of the measures clearly indicate that $N_E(8)$ is noticably worse than $N(24)$ and $N_E(4)$, which themselves have much closer averages.

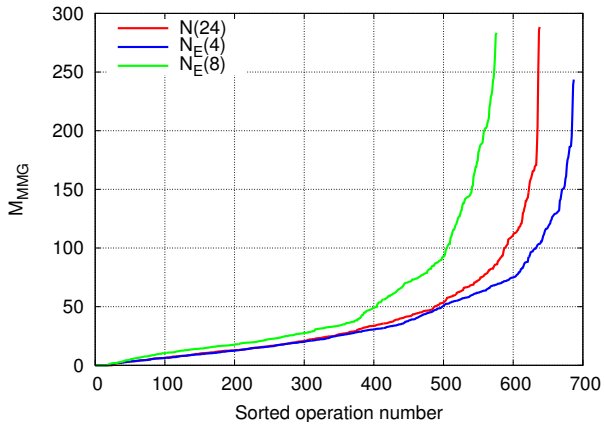


Figure 7.23: Maximum detail threshold comparison for sorted values of the Minimise Motion Groups measure $\mathcal{M}_{\mathrm{MMG}}$.

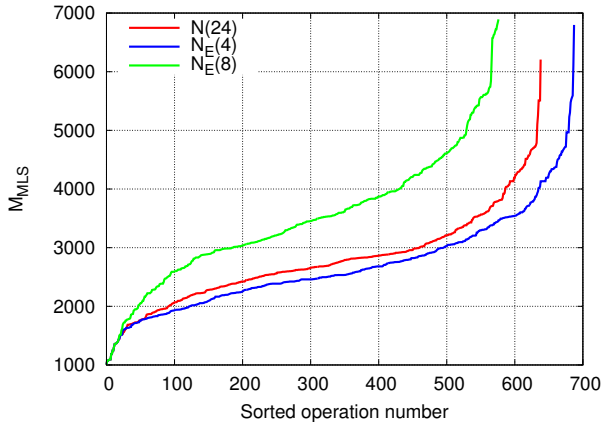


Figure 7.24: Maximum detail threshold comparison for sorted values of the Minimise Layout Size measure $\mathcal{M}_{\mathrm{MLS}}$.

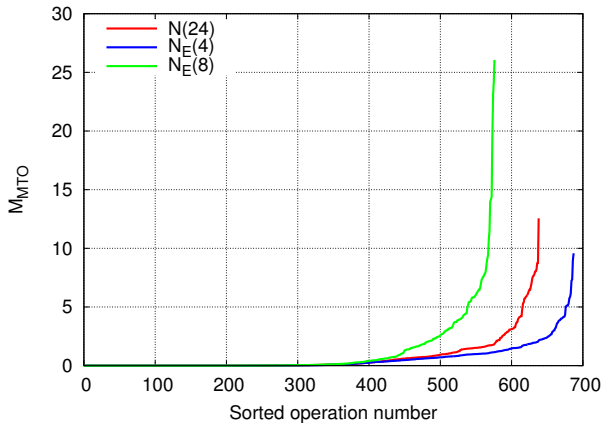


Figure 7.25: Maximum detail threshold comparison for sorted values of the Minimise Transient Occlusions measure $\mathcal{M}_{\mathrm{MTO}}$.

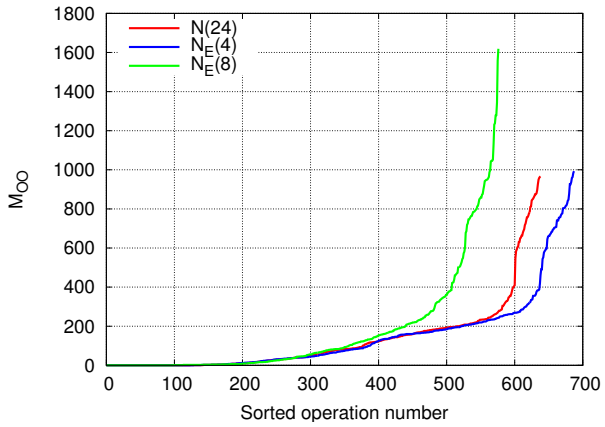


Figure 7.26: Maximum detail threshold comparison for sorted values of the Orthogonal Ordering measure $\mathcal{M}_{\mathrm{OO}}$.

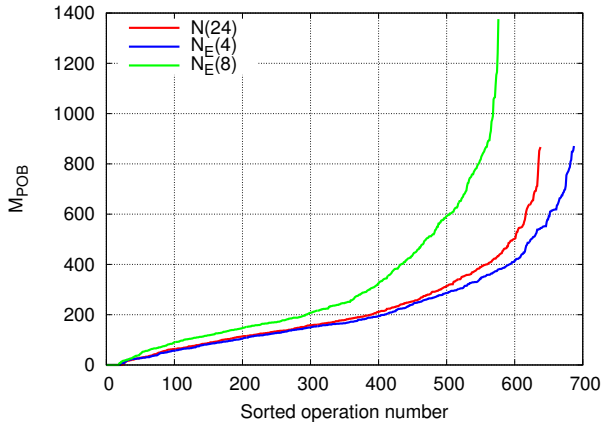


Figure 7.27: Maximum detail threshold comparison for sorted values of the Preservation of Bends measure $\mathcal{M}_{\mathrm{POB}}$.

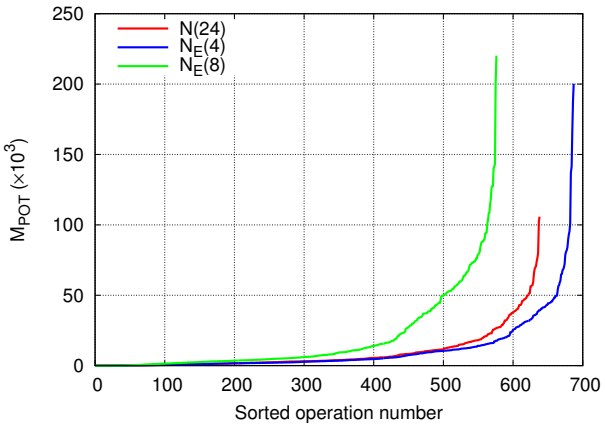


Figure 7.28: Maximum detail threshold comparison for sorted values of the Preservation of Topology measure $\mathcal{M}_{\mathrm{POT}}$.

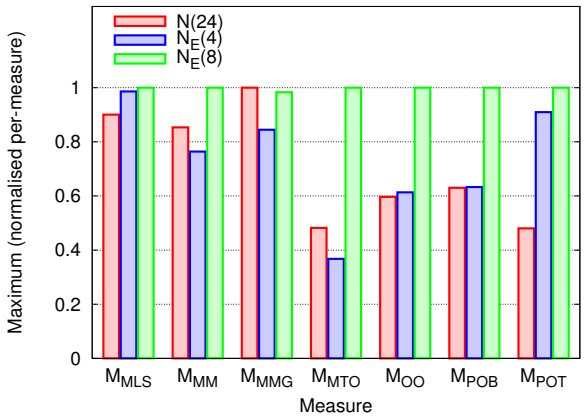


Figure 7.29: Maximum detail threshold comparison by using the normalised maximum of each measure.

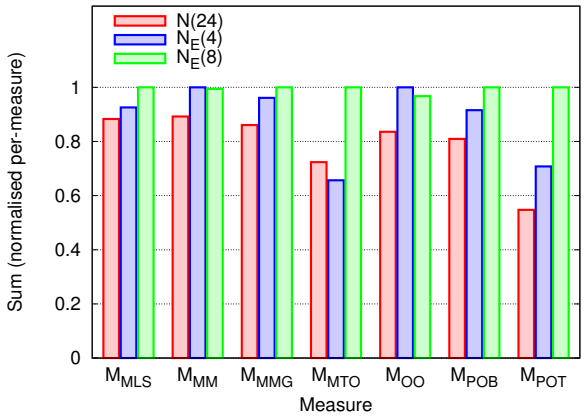


Figure 7.30: Maximum detail threshold comparison by using the normalised sum of each measure.

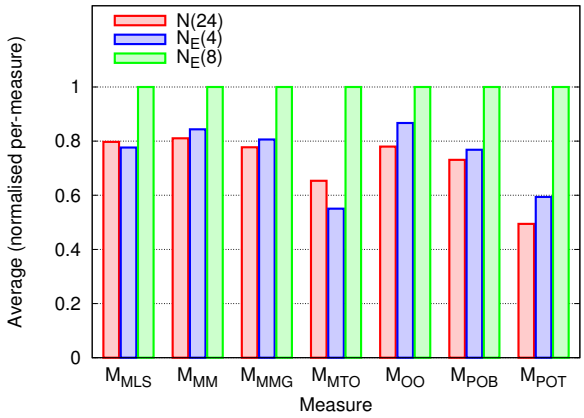


Figure 7.31: Maximum detail threshold comparison by using the normalised average of each measure.

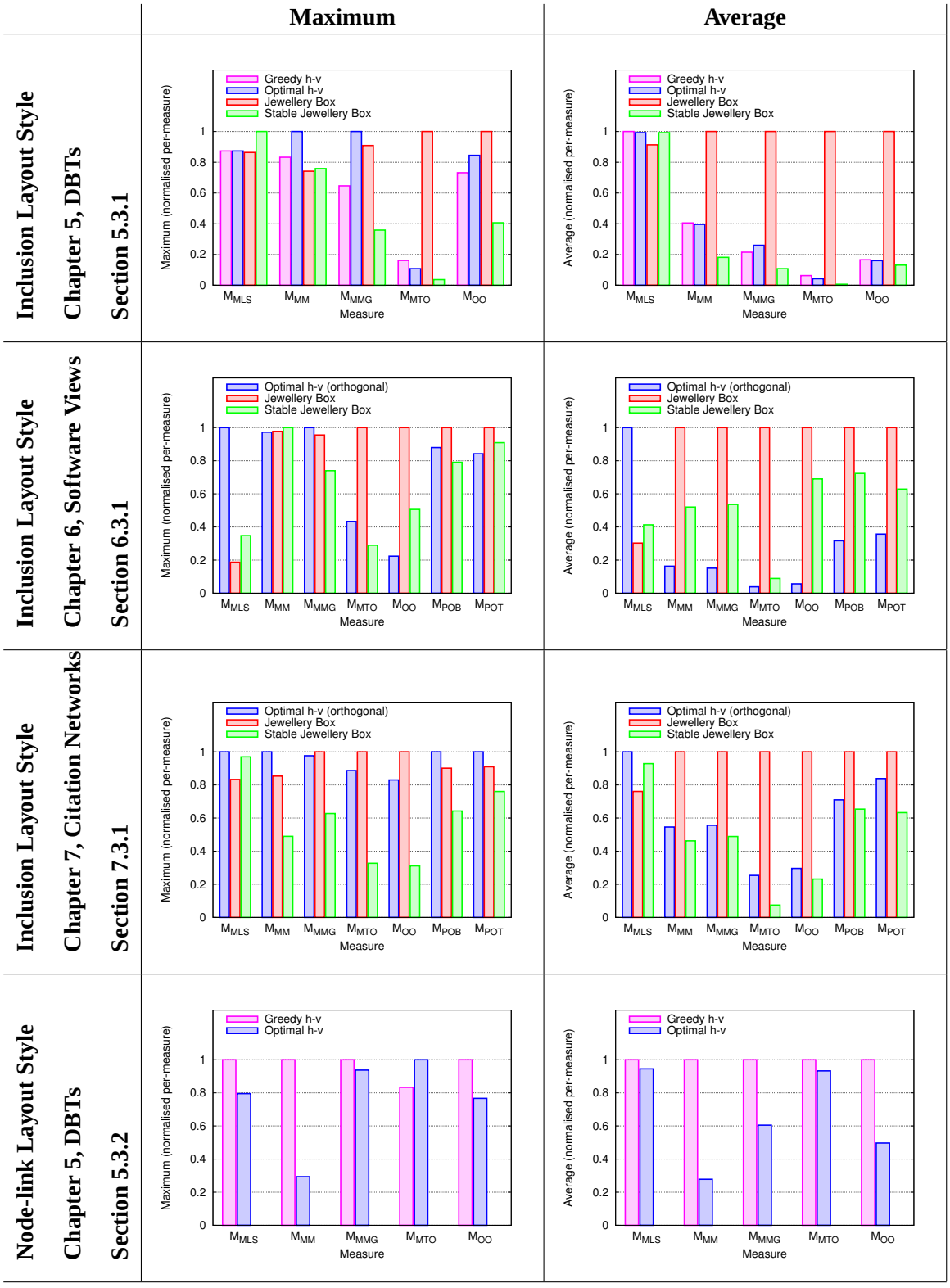


Figure 8.1: Summary of layout algorithm results, from Figures 5.10, 5.12, 6.12, 6.14, 7.9, 7.11, 5.18 and 5.20.

CHAPTER 8

Discussion

This chapter discusses the results of the three empirical investigations presented in Chapters 5, 6 and 7.

8.1 Layout algorithm

Layout algorithm results are presented for both inclusion and node-link layout styles in Sections 5.3.1, 5.3.2, 6.3.1 and 7.3.1. Figure 8.1 shows a summary of the results from these sections.

Observe that for layout size ($\mathcal{M}_{\mathrm{MLS}}$), the h-v based algorithms perform poorly for cluster hierarchies with large degrees (Sections 6.3.1 and 7.3.1), but perform comparatively well for smaller degrees (Section 5.3.1). The greedy h-v layout performs slightly worse than the optimal h-v layout (Section 5.3.2), which confirms its expected performance. In addition, the comparison between inclusion and node-link h-v layout algorithms confirms that there is no substantial difference between layout in the inclusion and node-link styles. By contrast, the JBILA produces the most compact layouts, performing at least as well as the h-v layout algorithms. This confirms our intuition, as the JBILA is designed to minimise the size of the layout without regard for any other factors. The performance of the SJBILA is between the JBILA and h-v based layout algorithms.

However, the JBILA performs very poorly in all the other measures ($\mathcal{M}_{\mathrm{MM}}$, $\mathcal{M}_{\mathrm{MMG}}$, $\mathcal{M}_{\mathrm{MTO}}$, $\mathcal{M}_{\mathrm{OO}}$, $\mathcal{M}_{\mathrm{POB}}$ and $\mathcal{M}_{\mathrm{POT}}$). In these measures, both the SJBILA and h-v based algorithms perform noticably better than the JBILA. In some cases the SJBILA outperforms h-v layout, and in others the h-v layout outperforms the SJBILA. This indicates that the SJBILA and h-v layout are both superior to the JBILA for Structural Zooming. For clustered hierarchies with large degrees, the SJBILA is superior due to its smaller layout.

Thus, it is concluded that the choice of layout algorithm is important in Structural Zooming, and has direct bearing on the performance of Structural Zooming. The difference between good

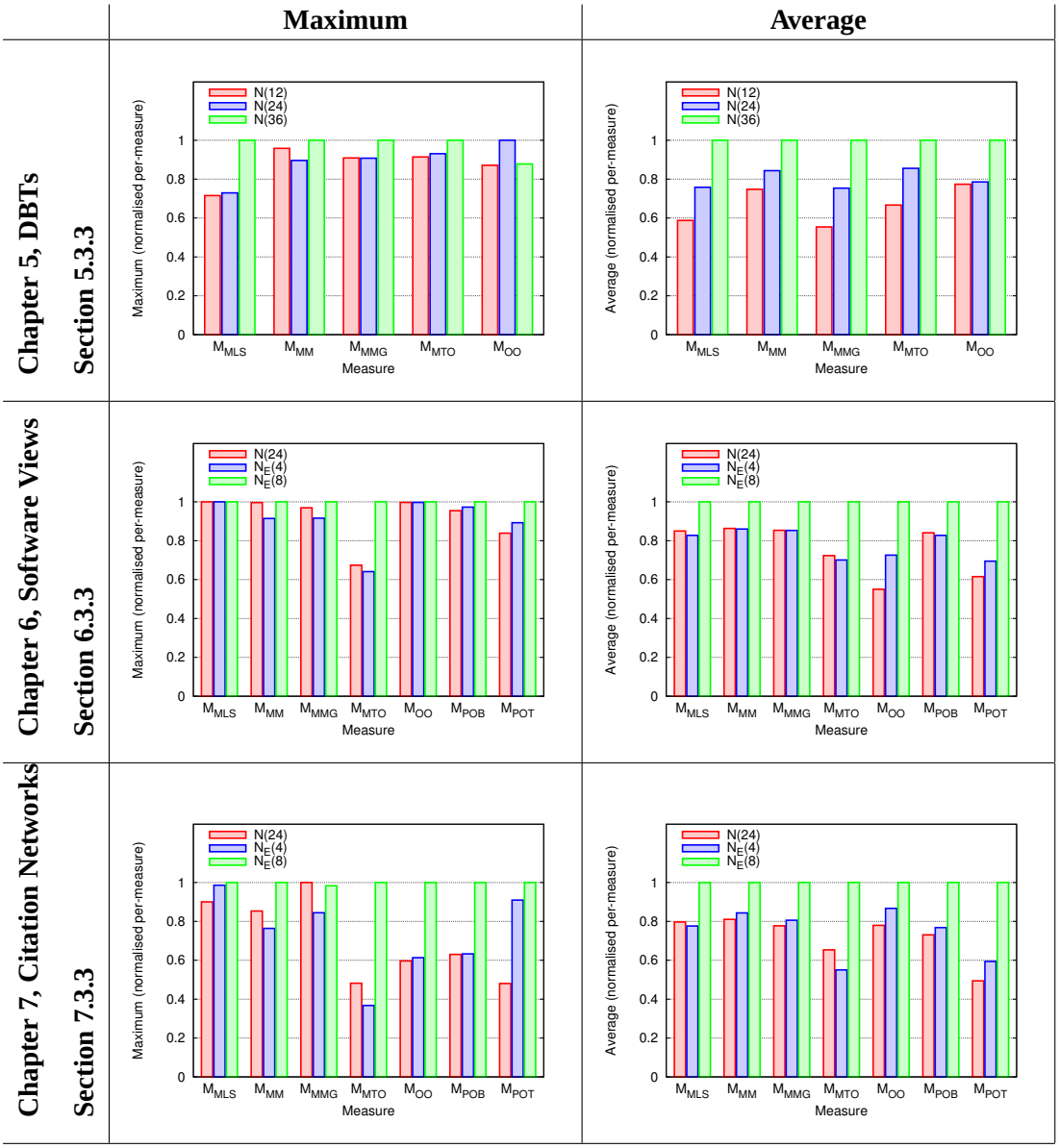


Figure 8.2: Summary of maximum detail threshold results, from Figures 5.26, 5.28, 6.32, 6.34, 7.29 and 7.31.

and bad quality layouts can be very large, not only in the traditional static layout measure of the layout size but also in the other dynamic measures. The results validate the SJBILA as a useful layout algorithm, particularly for Structural Zooming. This is because it is able to combine good layout size with good dynamic animation properties.

8.2 Maximum detail threshold

Results for the maximum detail threshold are presented in Sections 5.3.3, 6.3.3 and 7.3.3. Figure 8.2 shows a summary of the results from these sections.

These results confirm our intuition that the choice of maximum detail threshold influences the measures: more data content results in measures that are generally worse. For the $\mathcal{M}_{\mathrm{MM}}$, $\mathcal{M}_{\mathrm{MMG}}$, $\mathcal{M}_{\mathrm{MLS}}$ and $\mathcal{M}_{\mathrm{MTO}}$ measures, the DBT results (Section 5.3.3) seem to indicate that this relationship may be approximately linear. However, the rest of the results indicate that the relationship increases dramatically if the data content is allowed to rise too far. In particular, the results in Sections 6.3.3 and 7.3.3 indicate that the $N_E(8)$ maximum detail threshold is too high, causing measure values that are noticably higher than $N(24)$ and $N_E(4)$.

This suggests that there is a "window" of suitable or appropriate maximum detail threshold choices for a given set of data, and once this limit is exceeded the visualisation quality decreases rapidly. This supports the concept of constant visual complexity presented in Section 3.1, that is, that maintaining an optimal amount of on-screen information is beneficial for the user (in terms of the measures used here). In addition, since the measures are geometry-based but the maximum detail threshold is data-based, these results justify the monotone visual complexity assumption and support limiting the visual complexity by limiting the underlying data content. This is useful as it allows constant visual complexity visualisation methods to be simplified, as they need not be concerned with the exact visual content and may instead concentrate on the data content. However, the problem of determining an *appropriate* window of visual complexity or data content has not been addressed. This is a necessary step in ensuring the practical effectiveness of constant visual complexity in Structural Zooming techniques, and remains an important open problem.

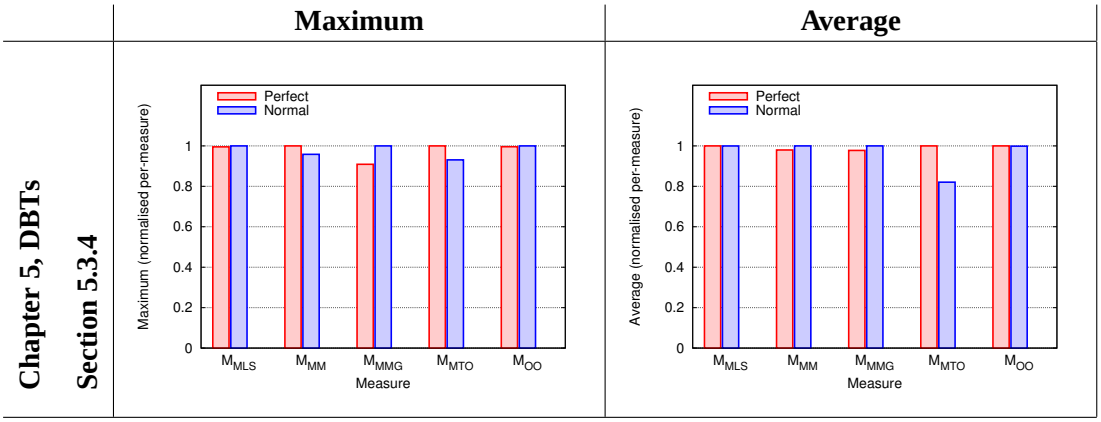


Figure 8.3: Summary of results of target node navigation strategies, from Figures 5.34 and 5.36.

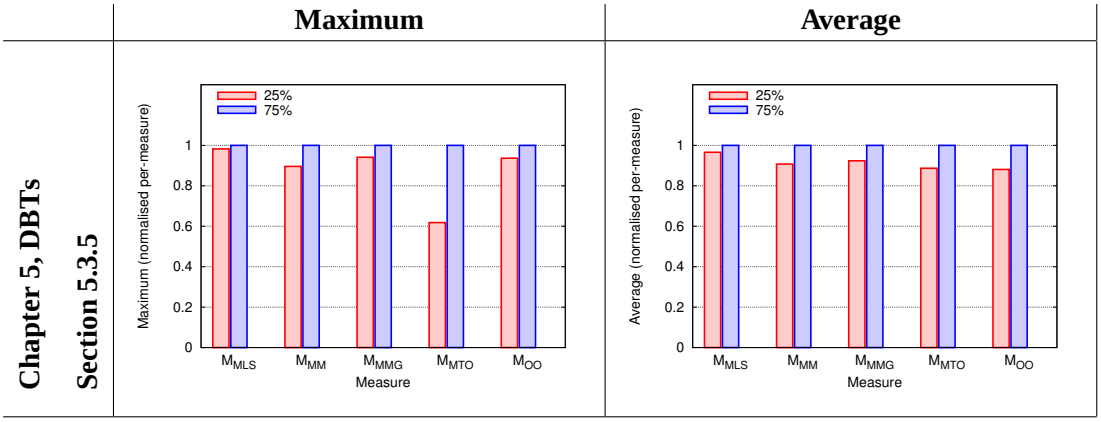


Figure 8.4: Summary of results of navigation strategy sample sizes, from Figures 5.42 and 5.44.

8.3 Navigation paths

Results for the comparison of target node navigation strategies is presented in Section 5.3.4. Figure 8.3 shows a summary of the results from this section.

These results show very little difference between the Perfect and Normal navigation strategies in both maximum values and average values. This is despite more operations being performed by the Normal navigation strategy, as shown by Figures 5.29 to 5.33.

This means that the mistake model used by the Normal navigation strategy does not cause large changes in the measures, and suggests that the two strategies are mostly equivalent. This justifies the choice to use only the Perfect navigation strategy in the subsequent empirical investigations in Chapters 6 and 7. It is concluded that the Perfect navigation strategy is useful as a deterministic benchmark navigation strategy, making the randomised approach of the Normal navigation strategy unnecessary.

One unexpected result is the average value for transient occlusions ($\mathcal{M}_{\mathrm{MTO}}$) is lower for the Normal navigation strategy. The most likely explanation for this is that some of the mistakes made by the Normal strategy turn out to be beneficial for subsequent target nodes, causing some occlusions to be avoided because some of the nodes are already expanded.

Results for the comparison of sample sizes is presented in Section 5.3.5. Figure 8.4 shows a summary of the results from this section.

These results also appear quite similar to each other, with the values for sample sizes of 75% being slightly larger than those for 25%. The difference in average values is small and approximately the same for all the measures. This suggests that the datasets are reasonably uniformly distributed, and that the results do not depend on the size of the region explored but rather on the fact that some random coverage is obtained. This suggests that the choice between 25% and 75% sample sizes is largely irrelevant in a study such as this, justifying the choice to use only a sample size of 25% (and smaller, in some cases), in the subsequent empirical investigations in Chapters 6 and 7.

An outlier in these results is the maximum value for transient occlusions ($\mathcal{M}_{\mathrm{MTO}}$) is noticably higher for 75%, whilst the average value follows the same trend as the other measures. This suggests that there may exist rare occasions with very large occlusions that are more likely to occur with larger sample sizes. In these cases using 25% is still preferable to 75%, since such pathological cases are less likely to occur while good coverage is still obtained.

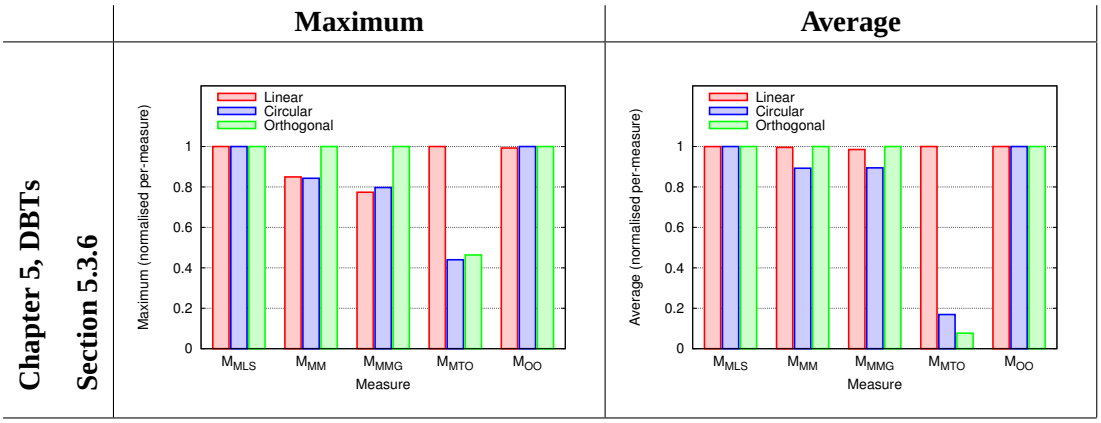


Figure 8.5: Summary of h-v node rotation results, from Figures 5.50 and 5.52.

8.4 H-V node rotation strategy

Results for the node rotation strategy for both inclusion and node-link layout styles is presented in Section 5.3.6. Figure 8.5 shows a summary of the results from this section.

These results show that there is no difference between linear, circular and orthogonal rotation in the layout size ($\mathcal{M}_{\mathrm{MLS}}$) and orthogonal ordering ($\mathcal{M}_{\mathrm{OO}}$) measures. This is intuitively expected and easily understood, based on the design of the rotation strategies and how they relate to these two measures.

However, it is immediately apparent that linear rotation is particularly bad in terms of transient occlusions ($\mathcal{M}_{\mathrm{MTO}}$). Circular rotation is not as good as orthogonal rotation, but is also much better than linear interpolation. The tradeoff is that orthogonal rotation is somewhat worse in terms of the motion measures ($\mathcal{M}_{\mathrm{MM}}$ and $\mathcal{M}_{\mathrm{MMG}}$). Interestingly, circular rotation performs better than linear rotation in the motion measures, where the opposite would be intuitively expected. Nevertheless, the orthogonal node rotation scheme is considered to be the most appropriate for the Structural Zooming of h-v layouts, as the benefits of few occlusions outweigh the additional motion it requires.

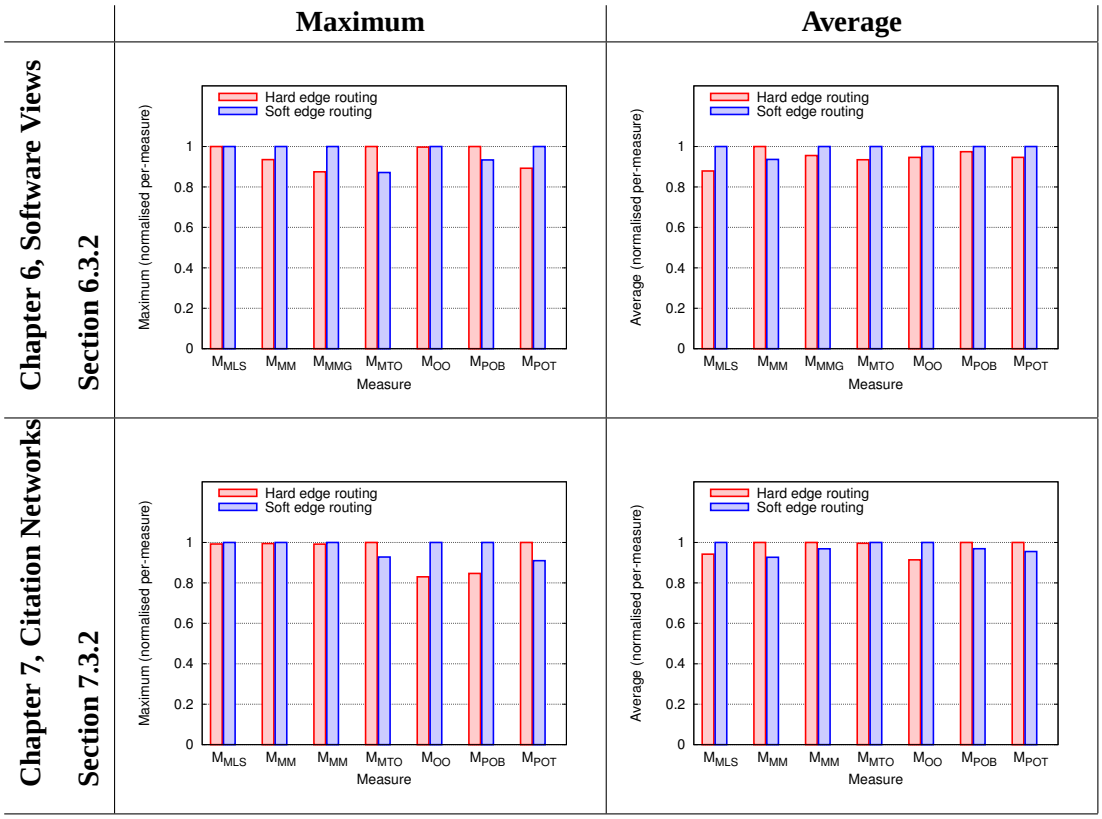


Figure 8.6: Summary of results of orthogonal edge routing — hard vs soft, from Figures 6.22, 6.24, 7.19 and 7.21.

8.5 Orthogonal edge routing — hard vs soft

Hard vs soft orthogonal edge routing results are presented in Sections 6.3.2 and 7.3.2. Figure 8.6 shows a summary of the results from these sections.

These results are clear — there is very little difference between the choice of hard and soft orthogonal edge routing in any of the experimental cases, for each of the measures. There is the possibility that there is a difference between them in other measures. Most notably, soft edge routing aims to maintain a minimum spacing between parallel edge segments, a property that is not captured by any of the empirical measures. However, from the results obtained, it is concluded that in these measures, there is no substantial difference between either of the edge routing strategies, even for the edge-based measures (Preservation of Bends $\mathcal{M}_{\mathrm{POB}}$ and Topology $\mathcal{M}_{\mathrm{POT}}$). This means that, based on its attempt to preserve edge spacing, it is likely that soft edge routing is preferable to hard edge routing. These results allow the choice of soft edge routing in the knowledge that the empirical measures will not be substantially worse as a result.

CHAPTER 9

Conclusion

This thesis presents *Structural Zooming*, a *data-driven* Focus + Context technique. This technique has all the advantages of contemporary geometric distortion based Focus + Context techniques, such as the Graphical Fisheye View and Hyperbolic Browser, but avoids the problems associated with these techniques. In particular, Structural Zooming maintains an approximately constant level of visual complexity, preserves spatial properties by virtue of the lack of any distortion, and leverages existing techniques. In addition, the benefits of geometric zooming are retained, including a high detail focus region and low detail context, smooth animation of transitions during user navigation, and preservation of a high quality and aesthetically pleasing data layout.

Structural Zooming has been presented in a general fashion, where it is applicable to any visualisation technique, and in the specific case of visualising relational data — namely, trees and clustered graphs. This latter method has been experimentally evaluated using data from three application areas and a series of empirical quality measures, derived from perceptual psychology, good design principles, and experience and intuition with the system. The findings confirm the usefulness of Structural Zooming as it is applied to relational data in this study.

9.1 Future research

This section briefly describes several possible directions for future research related to this thesis.

HCI user study: A full HCI user study would be able to investigate the psychophysical parameters of Structural Zooming. The results of such a study may assist with predicting actual human performance when using Structural Zooming.

Concurrent vs. consecutive animations: The question of how best to animate a collection of composable animations remains unanswered. This is effectively

the issue of how to partition the animations into groups of concurrent animations, and the order in which the groups should be consecutively animated. It also includes any related human-perception issues. This is mentioned in Section 3.3.

Efficient removal of non-monotonicities: The algorithm for removing non-monotonicities presented in Section 4.3.4 is fairly simple; however, it potentially requires repeated linear searches through the edge layout. In addition, it is not guaranteed to find transitions that are free from ambiguities when edges are non-monotone in both x and y. It would be interesting to find an alternative search strategy for the efficient removal of non-monoticities in edge layouts.

Improved animation strategy for SJBILA layout changes: The Stable Jewellery Box Inclusion Layout Algorithm (Section 4.2.1) does a good job of preserving the mental map and avoiding occlusions during transitions between layouts. However, it still uses linear interpolation, which can almost certainly be improved upon in order to further reduce occlusions during transitions. Such a system may make use of the existing Delaunay triangulations from the layout construction, or may incorporate elements of route planning from the field of robotics.

Arbitrary (non-orthogonal) edge layouts: The generalisation of orthogonal edge routing and animation (Section 4.3) to arbitrarily placed polylines (or smooth curves) would be quite challenging.

Structural Zooming of non-relational data: This thesis presents the general technique of Structural Zooming, as well as its application to relational data of trees and clustered graphs. Applying Structural Zooming to other types of information visualisation could yield good results and useful systems.

Clustered graph algorithms: The general strategy of treating clustered graphs as inclusion trees combined with node-link edges can limit the quality of the overall layout. An improvement would be the use of a clustered graph algorithm that is specifically designed to draw clustered graphs nicely.

Application to graphs: Structural Zooming may be applied to normal (non-clustered) graphs if a clustering algorithm is used to automatically cluster the nodes of the graph. Such clustering algorithms may be graph-theoretic or geometrically based, and may compute the clustering statically (that is, compute the clustering for the entire dataset *a priori*) or dynamically (that is, compute the clustering for the zoomed dataset during the user's navigation through the dataset).

Allow modification and interaction with the dataset: The Structural Zooming system implemented is a *browser*, in that it does not support modification of the overall dataset. The usefulness of Structural Zooming would be increased if it could be used to edit or interact with the data, instead of merely displaying it.

Bibliography

[1] Abello, J. and Korn, J., "Visualizing massive multi-digraphs," *Proceedings of IEEE Symposium on Information Visualization, 2000. InfoVis 2000.*, 2000, pp. 39–47. 1

[2] Abello, J., Korn, J., and Kreuseler, M., "Navigating giga-graphs," *Proceedings of the 6th Conference on Advanced Visual Interfaces*, May 2002, pp. 290–299, Trento, Italy. 1

[3] Abello, J. and Vitter, J., *External Memory Algorithms*, Volume 50 of the AMS-DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 1999. 1

[4] Aho, A. V., Sethi, R., and Ullman, J. D., *Compilers: Principles, Techniques and Tools*, Addison-Wesley, Reading, Massachusetts, 1986. 2.1.1

[5] Aigner, M., *Combinatorial Theory*, Springer Verlag, New York, 1979. 2.1.1, 2

[6] Association for Computing Machinery, "*ACM Computing Classification System*," `http://www.acm.org/class/`. 7.2

[7] Association for Computing Machinery, "*ACM Digital Library*," `http://portal.acm.org/dl.cfm`. 7.1

[8] Astrophysical Research Consortium, "The Sloan Digital Sky Survey Project Book," 1999, `http://www.astro.princeton.edu/PBOOK/`. 1

[9] Barabasi, A. and Albert, R., "Emergence of scaling in random networks," *Science*, Vol. 286, 1999, pp. 509–512. 7.1

[10] Baratz, A. E., *Algorithms for integrated circuit signal routing*, PhD thesis, Dept. of EECS, MIT, Cambridge, 1981. 2.1.2

[11] Bartram, L., Henigman, F., and Dill, J., "Intelligent zoom as metaphor and navigation tool in a multi-screen interface for network control systems," *Proceedings of IEEE International Conference on Systems, Man and Cybernetics '95*, Vol. 4, 1995, pp. 3122–3127. 1.2.4

[12] Bartram, L., Ho, A., Dill, J., and Henigman, F., "The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces," *Proceedings of Conference on User Interface Software and Technology (UIST) '95*, ACM Press, Nov. 1995, pp. 207–215. 1.2.4, 1.24

[13] Bartram, L., Ovans, R., Dill, J., Dyck, M., Ho, A., and Havens, W. S., "Contextual assistance in user interfaces to complex, time-critical systems: The intelligent zoom," *Proceedings of Graphics Interface (GI) '94*, 1994, pp. 216–224. 1.2.4

[14] Bartram, L., Uhl, A., and Calvert, T., "Navigating Complex Information with the ZTree," *Proceedings of Graphics Interface (GI)*, May 2000, pp. 11–18. 1.2.4

[15] Bartram, L. and Ware, C., "Filtering and brushing with motion," *Information Visualization*, Vol. 1, No. 1, March 2002, pp. 66–79. 1.2.5

[16] Bartram, L. R., *Enhancing Information Visualization with Motion*, PhD thesis, School of Computing Science, Simon Fraser University, 2001. 1.2.5

[17] Bassili, J., "Temporal and spatial contingencies in the perception of social events," *Journal of Personality and Social Psychology*, Vol. 33, No. 6, 1976, pp. 680–685. 4.2.1, 4.4, 4.4.5

[18] Baudisch, P., Good, N., and Stewart, P., "Focus plus context screens: combining display technology with visualization techniques," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'02)*, 2002, pp. 31–40. 1.2.3

[19] Bederson, B., Meyer, J., and Good, L., "Jazz: an extensible zoomable user interface graphics toolkit in java," *Proceedings of User Interface and Software Technology (UIST 2000)*, ACM Press, 2000, pp. 171–180. 1.2.1

[20] Bederson, B. B., "Fisheye menus," *Proceedings of ACM Conference on User Interface Software and Technology (UIST)*, 2000, pp. 217–225. 1.2.3

[21] Bederson, B. B. and Boltman, A., "Does animation help users build mental maps of spatial information?" *Proceedings of 1999 IEEE Symposium on Information Visualization (Info Vis '99)*, 1999, pp. 28–35. 1.2.5

[22] Bederson, B. B., Grosjean, J., and Meyer, J., "Toolkit design for interactive structured graphics," *IEEE Transactions on Software Engineering*, Vol. 30, No. 8, 2004, pp. 535–546. 1.2.1

[23] Böhringer, K.-F. and Paulisch, F. N., "Using constraints to achieve stability in automatic graph layout algorithms," *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'90)*, ACM Press, 1990, pp. 43–51. 4.2.1, 4.4, 4.4.2, 4.4.6

[24] Bowman, I. T., Holt, R. C., and Brewster, N. V., "Linux as a case study: Its extracted software architecture," *Proceedings of the 21st International Conference on Software Engineering (ICSE)*, ACM Press, 1999, pp. 555–563. 6.2

[25] Card, S. K., MacKinlay, J. D., and Shneiderman, B., *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann Series in Interactive Technologies, Academic Press, 1999. 58, 93

[26] Card, S. K., Pirolli, P., and Mackinlay, J. D., "The cost-of-knowledge characteristic function: Display evaluation for direct-walk information visualizations," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, 1994, pp. 238–244. 3.2

[27] Cockburn, A. and Savage, J., "Comparing speed-dependent automatic zooming with traditional scroll, pan, and zoom methods," *People and Computers XVII: British Computer Society Conference on Human Computer Interaction, Bath, England*, 2003, pp. 87–102. 1.2.1

[28] Cooperative Association for Internet Data Analysis (CAIDA), San Diego Supercomputer Center (SDSC), University of California, San Diego (UCSD), "Walrus — Graph Visualization Tool," `http://www.caida.org/tools/visualization/walrus/`. 1.2.3

[29] Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, MIT Press, Massachusetts, 1990. 2.1.1, 2.1.2

[30] de Resende, P. J., Lee, D. T., and Wu, Y. F., "Rectilinear shortest paths with rectangular barriers," *Proceedings of the First Annual Symposium on Computational Geometry*, ACM Press, New York, NY, USA, 1985, pp. 204–213. 4.3.4

[31] Dean, T. R., Malton, A. J., and Holt, R., "Union Schemas as a Basis for a C++ Extractor," *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE)*, IEEE Computer Society Press, 2001, pp. 59–67. 6.2

[32] di Battista, G., Eades, P., Tamassia, R., and Tollis, I. G., *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, New Jersey, 1999. 2.1.2, 3.1, 4.2, 4.4, 4.4.1, 4.4.3, 4.4.6, 4.4.7, 7.2

[33] Dill, J., Bartram, L., Ho, A., and Henigman, F., "A continuously variable zoom for navigating large hierarchical networks," *Proceedings of IEEE International Conference on Systems, Man and Cybernetics '94*, Vol. 1, 1994, pp. 386–390. 1.2.4

[34] Driver, J., MacLeod, P., and Dienes, Z., "Motion coherence and conjunction search: Implications for guided search theory," *Perception and Psychophysics*, Vol. 51, No. 1, 1991, pp. 79–85. 4.2.1, 4.4.5

[35] Dromey, R. G., personal communication, 1998. 5.1, 5.2, 5.1, 5.3, 5.2

[36] Dromey, R. G., "From requirements to design: Formalizing the key steps (invited keynote address)," *Proceedings of First International Conference on Software Engineering and Formal Methods*, 2003, pp. 2–11. 2.3.1, 5.1

[37] Dromey, R. G., "Using behavior trees to model the autonomous shuttle system," *Proceedings of 3rd International Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'04)*, 2004. 2.3.1, 5.1

[38] Dwyer, T. and Eades, P., "Visualising a fund manager flow graph with columns and worms," *Proceedings of the 6th International Conference on Information Visualisation, IV02*, IEEE Computer Society, 2002, pp. 147–158. 2.1.2

[39] Dwyer, T. and Eckersley, P., *Graph Drawing Software*, chap. WilmaScope — A 3D Graph Visualization System, Springer, Berlin, 2003, pp. 55–75. 1.2.4

[40] Eades, P., "A heuristic for graph drawing," *Congressus Numerantium*, Vol. 42, 1984, pp. 149–160. 4.2

[41] Eades, P., Cohen, R., and Huang, M., "Online animated graph drawing for web navigation," *Proceedings of Graph Drawing 1997*, Vol. 1353 of *Lecture Notes in Computer Science*, Springer Verlag, 1997, pp. 330–335. 3.3

[42] Eades, P. and Lai, W., "Algorithms for disjoint node images," *Proceedings of the 15th Australian Computer Science Conference (ACSC)*, 1992, pp. 253–265. 1.1, 4.2

[43] Eades, P., Lai, W., Misue, K., and Sugiyama, K., "Preserving the mental map of a diagram," *Proceedings of Compugraphics '91*, 1991, pp. 34–43. 1.1, 3.3

[44] Eades, P., Lin, T., and Lin, X., "Two tree drawing conventions," *International Journal of Computational Geometry and Applications*, Vol. 3, No. 2, 1993, pp. 133–153. 2.1.1, 2.2.1, 2.2.1, 2.2.1, 2.2.1, 2.2.2

[45] Estivill-Castro, V. and Houle, M. E., "Robust distance-based clustering with applications to spatial data mining," *Algorithmica*, Vol. 30, No. 2, 2001, pp. 216–242. 4.4, 4.4.5

[46] Farquhar, A., Fikes, R., and Rice, J., *The Ontolingua Server: A Tool for Collaborative Ontology Construction*, Stanford Knowledge Systems Laboratory, *KSL Technical Report 96-26*, 1996, `http://ontolingua.stanford.edu/`. 2.3.1

[47] Feng, Q., *Algorithms for Drawing Clustered Graphs*, PhD thesis, University of Newcastle, 1997. 2.1.2

[48] Finnigan, P., Holt, R. C., Kalas, I., Kerr, S., Kontogiannis, K., Muller, H., Mylopoulos, J., Perelgut, S., Stanley, M., and Wong, K., "The software bookshelf," *IBM Systems Journal*, Vol. 36, No. 4, November 1997, pp. 564–593. 6.2

[49] Fortune, S., "Voronoi diagrams and delaunay triangulations," *Computing in Euclidean Geometry*, edited by D.-Z. Du and F. Hwang, World Scientific, Singapore, 1992. 2.2.2

[50] Free Software Foundation, "*The GNU 'hello' program*," `http://www.gnu.org/software/hello/`. 6.2

[51] Free Software Foundation, "*The GNU Project*," `http://www.gnu.org/`. 6.2

[52] Free Software Foundation, *The GNU Bash Reference Manual*, Free Software Foundation, Boston, MA, 2002, `http://www.gnu.org/software/bash/`. 6.2

[53] Free Software Foundation, *The GNU Grep Reference Manual*, Free Software Foundation, Boston, MA, 2002, `http://www.gnu.org/software/grep/`. 6.2

[54] Friedrich, C., *Animation in Relational Information Visualization*, PhD thesis, University of Sydney, 2002. 1.2.5, 3.3, 4.2, 4.2.1, 4.4, 4.4.3, 4.4.4, 4.4.6

[55] Friedrich, C. and Eades, P., "Graph drawing in motion," *Journal of Graph Algorithms and Applications*, Vol. 6, No. 3, 2002, pp. 353–370. 1.2.5, 4.2, 4.4, 4.4.3, 4.4.4, 4.4.6

[56] Fruchterman, T. M. J. and Reingold, E. M., "Graph drawing by force-directed placement," *Software - Practice and Experience*, Vol. 21, No. 11, 1991, pp. 1129–1164. 4.2

[57] Funkhouser, T. A. and Séquin, C. H., "Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments," *Proceedings of SIGGRAPH '93*, 1993, pp. 247–254. 3.2

[58] Furnas, G. W., "The fisheye view: a new look at structured files," Bell Laboratories Technical Memorandum 81-11221-9, October 12, 1981. Published in [25], pp. 312–330. 1.2.1, 1.2.4

[59] Furnas, G. W., "Generalized fisheye views," *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'86)*, ACM Press, 1986, pp. 16–23. 1.2.1, 1.2.4, 1.22

[60] Furnas, G. W., "Effective view navigation," *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'97)*, ACM Press, 1997, pp. 367–374. 3.2

[61] Furnas, G. W. and Bederson, B. B., "Space-scale diagrams: Understanding multiscale interfaces," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'95)*, 1995, pp. 234–241. 1.2.1

[62] Gansner, E. R., Mocenigo, J. M., and North, S. C., "Visualizing software for telecommunication services," *Proceedings of the 2003 ACM symposium on Software visualization*, 2003, pp. 151–157. 7.2

[63] Goldberg, A., *Smalltalk-80: the interactive programming environment*, Addison-Wesley series in computer science, Addison-Wesley, Reading, MA, 1983. 3.2

[64] Goldberg, A. and Robson, D., *Smalltalk-80: the language and its implementation*, Addison-Wesley series in computer science, Addison-Wesley, Reading, MA, 1983. 3.2

[65] Goldstein, E. B., *Sensation and Perception*, Brooks/Cole, Pacific Grove, CA, 5th ed., 1997. 1.1

[66] Gosling, J., Joy, B., Steele, G., and Bracha, G., *The Java Language Specification*, Addison-Wesley, 2nd ed., 2000. 2.2.2

[67] Graham, S. L., Kessler, P. B., and McKusick, M. K., "gprof: a call graph execution profiler," *SIGPLAN Symposium on Compiler Construction*, 1982, pp. 120–126. 2.1.2

[68] Grochowski, E. and Halem, R. D., "Technological impact of magnetic hard disk drives on storage systems," *IBM Systems Journal*, Vol. 42, No. 2, July 2003, pp. 338–346. 1

[69] Gruber, T. R., "A translation approach to portable ontologies," *Knowledge Acquisition*, Vol. 5, No. 2, 1993, pp. 199–220. 2.3.1

[70] Harel, D., "On visual formalisms," *Communications of the ACM*, Vol. 31, No. 5, May 1988, pp. 514–530. 2.1.2

[71] Hoffman, D. D., *Visual intelligence: how we create what we see*, W. W. Norton, New York, 1998. 1.1

[72] Holeĉek, J. and Sojka, P., "Animations in pdfTeX-generated PDF: A New Method for Directly Embedding Animation into PDF," *Proceedings of International Conference on TeX, XML and Digital Typography (TUG 2004)*, 2004. 1.6

[73] Holt, R. C., "Software architecture abstraction and aggregation as algebraic manipulations," *Proceedings of the 1999 Conference of the IBM Centre for Advanced Studies on Collaborative Research*, 1999. 2.1.2, 6.2

[74] Hoppe, H., "Progressive meshes," *Proceedings of SIGGRAPH '96*, 1996, pp. 99–108. 3.2

[75] Horn, M., "Analysis and computation schemes for p-median heuristics," *Environment and Planning A*, Vol. 28, 1996, pp. 1699–1708. 4.4.5

[76] Huang, M. L., *Online Information Visualization of Huge Data Spaces*, PhD thesis, University of Newcastle, 1999. 2.1.2

[77] Huang, X. and Lai, W., "Force-transfer: A new approach to removing overlapping nodes in graph layout," *Proceedings of the 26th Australian Computer Science Conference (ACSC)*, 2003, pp. 349–358. 4.2

[78] Igarashi, T. and Hinckley, K., "Speed-dependent automatic zooming for browsing large documents," *Proceedings of User Interface and Software Technology (UIST 2000)*, ACM Press, 2000, pp. 139–148. 1.2.1

[79] "Inxight StarTree," `http://www.inxight.com/products/sdks/st/`. 1.19

[80] Itoh, T., Kajinaga, Y., Ikehata, Y., and Yamaguci, Y., "Data Jewelry-Box: A Graphics Showcase for Large-Scale Hierarchical Data Visualization," *IBM Research, TRL Research Report, RT0427*, 2002. 2.2.2

[81] Johnson, B. and Shneiderman, B., "Tree-maps: A space-filling approach to the visualization of hierarchical information structures," *Proc. IEEE Visualization '91*, IEEE, 1991, pp. 284–291. 2.2.1

[82] Keim, D. A., "Information visualization and visual data mining," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 7, No. 1, 2002, pp. 100–107. 1

[83] Kernighan, B. W. and Pike, R., *The UNIX Programming Environment*, Prentice Hall, New Jersey, 1984. 2.1.1

[84] Knuth, D. E., "Computer-drawn flowcharts," *Communications of the ACM*, Vol. 6, No. 9, Sept. 1963, pp. 555–563. 2.1.2

[85] Krasner, G. E. and Pope, S. T., "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, Vol. 1, No. 3, 1988, pp. 26–49. 3.2

[86] Krasner, G. E. and Pope, S. T., "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System," `http://www.create.ucsb.edu/~stp/PostScript/mvc.pdf`, 1988. 3.2

[87] Kuipers, B., "Modelling spatial knowledge," *Cognitive Science*, Vol. 2, 1978, pp. 129–153. 1.1

[88] Kuipers, B., "The 'map in the head' metaphor," *Environment and Behavior*, Vol. 14, 1982, pp. 202–220. 1.1

[89] Lai, W., *Building Interactive Diagram Applications*, PhD thesis, University of Newcastle, 1994. 1.1, 4.2

[90] Lai, W. and Eades, P., "A graph model which supports flexible layout functions," Tech. Rep. 96-15, University of Newcastle, Callaghan 2308, Australia, 1996. 1.1

[91] Lai, W. and Eades, P., "Removing edge-node intersections in drawings of graphs," *Information Processing Letters*, Vol. 81, No. 2, January 2002, pp. 105–110. 7.2

[92] Lamping, J., Rao, R., and Pirolli, P., "A focus+context technique based on hyperbolic geometry for visualizing large hierarchies," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'95)*, 1995, pp. 401–408. 1.2.3, 1.18

[93] Leung, Y. K. and Apperley, M. D., "A review and taxonomy of distortion-oriented presentation techniques," *ACM Transactions on Computer-Human Interaction*, Vol. 1, No. 2, 1994, pp. 126–160, Reprinted in [25]. 1.2.3, 1.12, 1.13, 1.14, 1.16, 1.3

[94] "LexisNexis Inxight StarTree view," `http://www.lexisnexis.com/startree/`. 1.19

[95] Mackinlay, J. D., Robertson, G. G., and Card, S. K., "The perspective wall: detail and context smoothly integrated," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'91)*, 1991, pp. 173–176. 1.2.3

[96] Martello, S. and Vigo, D., "Exact solution of the two-dimensional finite bin packing problem," *Management Science*, Vol. 44, No. 3, 1998, pp. 388–399. 2.1.1

[97] McLeod, P., Driver, J., and Crisp, J., "Visual search for a conjunction of movement and form is parallel," *Nature*, Vol. 332, 1988, pp. 154–155. 4.2.1, 4.4.5

[98] Michal, G., *Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology*, Wiley-Spektrum, 1999. 2.1.2

[99] Michotte, A., *The perception of causality*, Methuen, London, 1963. 4.2.1, 4.4, 4.4.5

[100] Miller, G. A., "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *Psychological Science*, Vol. 63, 1956, pp. 81–97. 1.1

[101] Misue, K., Eades, P., Lai, W., and Sugiyama, K., "Layout adjustment and the mental map," *Journal of Visual Languages and Computing*, Vol. 6, No. 2, 1995, pp. 183–210. 1.1, 3.3, 4.4, 4.4.2, 4.4.3

[102] Müller, H., Wong, K., and Tilley, S., "Understanding software systems using reverse engineering technology," *Proceedings of the 62nd Congress of L'Association Canadienne Francaise pour l'Avancement des Sciences (ACFAS)*, 1994. 2.1.2, 6.2

[103] Müller, H. A. and Klashinsky, K., "Rigi — a system for programming-in-the-large," *Proceedings of the 10th International Conference on Software Engineering*, ACM Press, 1988, pp. 80–86. 6.2

[104] Munzner, T., "Exploring Large Graphs in 3D Hyperbolic Space," *IEEE Computer Graphics and Applications*, Vol. 18, No. 4, July/August 1998, pp. 18–23. 1.2.3

[105] Nakayama, K. and Silverman, G., "Serial and parallel processing of visual feature conjunctions," *Nature*, Vol. 320, 1986, pp. 264–265. 4.2.1, 4.4.5

[106] Nelson, P. A., *bc, an arbitrary precision calculator language*, Free Software Foundation, Boston, MA, 1997, `http://www.gnu.org/software/bc/`. 6.2

[107] Nesbitt, K., *Designing multi-sensory metaphors for interacting with abstract data in virtual environments*, PhD thesis, University of Sydney, 2002. 1.1, 2.3.1, 3.1

[108] Noik, E. G., "Exploring large hyperdocuments: fisheye views of nested networks," *Proceedings of the fifth ACM conference on Hypertext*, 1993, pp. 192–205. 1.2.4

[109] Noik, E. G., "Exploring large hyperdocuments: fisheye views of nested networks," *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: distributed computing - Volume 2*, 1993, pp. 661–676. 1.2.4

[110] Noik, E. G., "Layout-independent fisheye views of nested graphs," *Proceedings IEEE Symposium on Visual Languages (VL)*, edited by E. P. Glinert and K. A. Olsen, IEEE Computer Society, 1993, pp. 336–341. 1.2.4, 1.23

[111] Noik, E. G., *Dynamic Fisheye Views: Combining Dynamic Queries and Mapping with Database Views*, PhD thesis, Department of Computer Science, University of Toronto, April 1996. 1.2.4

[112] O'Rourke, J., *Computational Geometry in C*, Cambridge University Press, New York, 2nd ed., 1998. 2.2.2, 7.2

[113] Paek, T., Dumais, S., and Logan, R., "Wavelens: A new view onto internet search results," *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'04)*, 2004, pp. 727–734. 1.2.3

[114] Parker, G., Franck, G., and Ware, C., "Visualization of Large Nested Graphs in 3D: Navigation and Interaction," *Journal of Visual Languages and Computing*, Vol. 9, No. 3, 1998, pp. 299–317. 1.2.4

[115] Pattison, T., Vernik, R., and Phillips, M., "Information visualisation using composable layouts and visual sets," *Australian symposium on Information visualisation*, 2001, pp. 1–10. 7.2

[116] Perlin, K. and Fox, D., "Pad: An alternative approach to the computer interface," *Proceedings of the 20th annual conference on Computer graphics (SIGGRAPH '93)*, ACM Press, 1993, pp. 57–64. 1.2.1

[117] Pizzini, K. and Stallman, R., *dc, an arbitrary precision calculator*, Free Software Foundation, Boston, MA, 1998, `http://www.gnu.org/software/bc/manual/dc-1.05/`. 6.2

[118] Plaisant, C., Carr, D., and Shneiderman, B., "Image-browser taxonomy and guidelines for designers," *IEEE Software*, Vol. 12, No. 2, March 1995, pp. 21–32. 1.2.2

[119] Plumlee, M. and Ware, C., "Zooming, Multiple Windows, and Visual Working Memory," *Proceedings of the 6th Conference on Advanced Visual Interfaces*, May 2002, Trento, Italy. 1.2.2

[120] Plumlee, M. and Ware, C., "An evaluation of methods for linking 3D views," *2003 Symposium on Interactive 3D Graphics*, 2003, pp. 193–201. 1.2.2, 1.11

[121] Plumlee, M. and Ware, C., "Integrating Multiple 3D Views through Frame-of-Reference Interaction," *Proceedings of the conference on Coordinated and Multiple Views In Exploratory Visualization*, 2003, pp. 34–43. 1.2.2

[122] Pulo, K., "AnimFig pdfLaTeX package," `http://www.kev.pulo.com.au/animfig/`. 1.6

[123] Pulo, K., Eades, P., and Takatsuko, M., "Smooth structural zooming of h-v inclusion tree layouts," *Proceedings of the First International Conference on Coordinated and Multiple Views In Exploratory Visualization*, 2003, pp. 14–25. 1.6

[124] Pulo, K. and Takatsuka, M., "Inclusion tree layout convention: An empirical investigation," *Proceedings of the Australian Symposium on Information Visualisation*, CRPIT Vol 24, Tim Pattison and Bruce Thomas, eds, 2003, pp. 27–35. 1.6, 2.3

[125] Purchase, H., "Which aesthetic has the greatest effect on human understanding?" *5th Intl. Symp. Graph Drawing (GD'97)*, Vol. 1353 of *Lecture Notes in Computer Science*, Springer-Verlag, 1997, pp. 248–261. 3.4, 4.3, 4.4, 4.4.1, 4.4.6, 4.4.7

[126] Pylyshyn, Z., Burkell, J., Fisher, B., Sears, C., Schmidt, W., and Trick, L., "Multiple parallel access in visual attention," *Canadian Journal of Experimental Psychology*, Vol. 48, No. 2, June 1994, pp. 260. 3.3, 4.2.1, 4.4, 4.4.5

[127] Quigley, A. J., *Large Scale Relational Information Visualization, Clustering, and Abstraction*, PhD thesis, University of Newcastle, 2001. 2.1.2

[128] Rao, R. and Card, S. K., "The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'94)*, 1994, pp. 318–322. 1.2.3

[129] Reingold, E. and Tilford, J., "Tidier drawing of trees," *IEEE Trans. Softw. Eng.*, Vol. 7, No. 2, 1981, pp. 223–228. 2.1.1

[130] Robertson, G. G. and Mackinlay, J. D., "The document lens," *ACM Symposium on User Interface Software and Technology (UIST)*, 1993, pp. 101–108. 1.15, 1.2.3

[131] Russell, S. J. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, 1995. 2.1.2

[132] Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, Massachusetts, 1990. 2.1.1

[133] Samet, H., *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, Massachusetts, 1990. 2.1.1

[134] San Diego Supercomputer Center, "High-Density Tiled Display," 2002, `http://vis.sdsc.edu/research/tileddisplay.html`. 1.1

[135] Sander, G., "*Layout of compound directed graphs*," Technical Report A/03/96, Universit at des Saarlandes, June 1996. 2.1.2

[136] Sarkar, M. and Brown, M. H., "Graphical fisheye views of graphs," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'92)*, 1992, pp. 83–91. 1.2.3, 1.17

[137] Sarkar, M. and Brown, M. H., "Graphical fisheye views," *Communications of the ACM*, Vol. 37, No. 12, December 1993, pp. 73–83. 1.2.3, 1.17

[138] Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S., and Roseman, M., "Navigating hierarchically clustered networks through fisheye and full-zoom methods," *ACM Transactions on Computer-Human Interaction (TOCHI)*, Vol. 3, No. 2, June 1996, pp. 162–188. 1.2.4

[139] Seckel, A., *The Art of Optical Illusions*, The Five Mile Press, 2000. 1.1

[140] Sekuler, R. and Blake, R., *Perception*, McGraw-Hill, 4th ed., 2002. 1.1

[141] Shannon, C. E., "A mathematical theory of communication," *Bell Systems Technical Journal*, Vol. 27, No. 3, July 1948, pp. 379–423. 3.1

[142] Sharir, M. and Schorr, A., "On shortest paths in polyhedral spaces," *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, ACM Press, 1984, pp. 144–153. 2.1.2

[143] Slay, H., Phillips, M., Thomas, B., and Vernik, R., "Keg master: a graph-aware visual editor for 3D graphs," *Proceedings of the Fourth Australian user interface conference on User interfaces*, 2003, pp. 99–103. 7.2

[144] Software Architecture Group, University of Waterloo, "The Software Architecture Toolkit," `http://swag.uwaterloo.ca/swagkit/`. 6.1, 6.2

[145] Software Architecture Group, University of Waterloo, "C488 PBS Software Bookshelf," 14 October 1998, `http://swag.uwaterloo.ca/pbs/examples/C488/`. 6.2

[146] Spence, R. and Apperley, M. D., "Database navigation: An office environment for the professional," *Behaviour and Information Technology*, Vol. 1, No. 1, 1982, pp. 43–54. 1.2.3

[147] Stallman, R. M. and the GCC Developer Community, *Using GCC: The GNU Compiler Collection Reference Manual*, GNU Press, Boston, MA, 2003, `http://gcc.gnu.org/`. 6.2

[148] Storey, M.-A. D., Fracchia, F. D., and Müller, H. A., "Customizing a fisheye view algorithm to preserve the mental map," *Journal of Visual Languages and Computing*, Vol. 10, 1999, pp. 245–267. 1.2.3

[149] Storey, M.-A. D. and Müller, H. A., "Graph layout adjustment strategies," *Proceedings of Graph Drawing 1995*, Lecture Notes in Computer Science, Springer Verlag, 1995, pp. 487–499. 1.1, 3.3

[150] Storey, M.-A. D., Wong, K., and Müller, H. A., "Rigi: A visualization environment for reverse engineering," *Proceedings of the 1997 International Conference on Software Engineering (ICSE)*, ACM Press, 1997, pp. 606–607. 6.2

[151] Sugiyama, K. and Misue, K., "Visualization of structural information: Automatic drawing of compound digraphs," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 4, 1991, pp. 876–892. 2.1.2

[152] Sugiyama, K., Tagawa, S., and Toda, M., "Methods for visual understanding of hierarchical systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 2, 1981, pp. 109–125. 2.2.2

[153] Sutherland, I. E., "Sketchpad: a man-machine graphical communication system," *AFIPS Conference Proceedings 23*, 1963, pp. 323–328. 1.1

[154] Sutherland, I. E., "Sketchpad: a man-machine graphical communication system," Technical Report, UCAM-CL-TR-574, University of Cambridge, September 2003, `http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-574.pdf`. 1.1

[155] Systä, T., "On the Relationships between Static and Dynamic Models in Reverse Engineering Java Software," *Proceedings of the 6th Working Conference on Reverse Engineering (WCRE99), Atlanta, GA, USA*, IEEE Computer Society Press, 1999, pp. 304–313. 6.1

[156] Teitz, M. B. and Bart, P., "Heuristic methods for estimating the generalised vertex median of a weighted graph," *Operations Research*, Vol. 16, 1968, pp. 955–961. 4.4.5

[157] Tolman, E. C., "Cognitive maps in rats and men," *Image and Environment, cognitive mapping and spatial behaviour*, edited by R. M. Downs and D. Stea, Edward Arnold, Chicago, 1948. 1.1

[158] Tom Sawyer Software Corporation, "Developer's Guide, Tom Sawyer Visualization, Version 6.0, Java Edition," 2004, `http://www.tomsawyer.com/`. 2.2.2, 2.2.3

[159] Tufte, E. R., *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut, 1983. 1, 3.1, 4.4

[160] Tufte, E. R., *Envisioning Information*, Graphics Press, Cheshire, Connecticut, 1990. 4.4

[161] Tufte, E. R., *Visual Explanations: Images and Quantities, Evidence and Narrative*, Graphics Press, Cheshire, Connecticut, 1997. 4.4

[162] Tversky, B., "Cognitive maps, cognitive collages, and spatial mental models," *Spatial information theory: a theoretical basis for GIS : European conference, COSIT'93*, edited by A. U. Frank and I. Campari, Springer-Verlag, New York, 1993, Marciana Marina, Elba Island, Italy, September 19-22, 1993. 1.1

[163] Ullman, J. D., *Principles of Database and Knowledge-Base Systems*, Computer Science Press, Rockville, Maryland, 1989. 2.1.1

[164] Uschold, M., King, M., Moralee, S., and Zorgios, Y., *The Enterprise Ontology*, Vol. 13, Special Issue on Putting Ontologies to Use (eds. M. Uschold. and A. Tate.) of The Knowledge Engineering Review, Stanford Knowledge Systems Laboratory, *KSL Technical Report 96-26*, 1998. 2.3.1

[165] van Wijk, J. J. and Nuij, W. A. A., "Smooth and efficient zooming and panning," *Proceedings of IEEE Symposium on Information Visualization (INFOVIS 2003)*, 2003, pp. 15–23. 1.7, 1.2.1

[166] Ware, C., *Information Visualization: Perception for Design*, Morgan Kaufmann Interactive Technologies Series, 1st ed., 2000. 1, 1.1, 1.1, 1.5, 1.2.5, 4.4

[167] Ware, C. and Bobrow, R., "Motion to support rapid interactive queries on node–link diagrams," *ACM Transactions on Applied Perception (TAP)*, Vol. 1, No. 1, July 2004, pp. 3–18. 1.2.5

[168] Ware, C. and Fleet, D., "Context sensitive flying interface," *1997 Symposium on Interactive 3D Graphics*, 1997, pp. 127–130. 1.2.1

[169] Ware, C. and Franck, G., "Evaluating stereo and motion cues for visualizing information nets in three dimensions," *ACM Transactions on Graphics (TOG)*, Vol. 15, No. 2, April 1996, pp. 121–140. 1.2.5

[170] Ware, C. and Lewis, M., "The DragMag image magnifier," *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'95)*, 1995, pp. 407–408. 1.2.2, 1.9

[171] Werfalli, J., "Review: ViewSonic VP2290b - High-Resolution TFT," 30 June 2004, Trusted-Reviews, `http://www.trustedreviews.com/article.aspx?art=533`. 1.1

[172] Wilson, R. J., *Introduction to Graph Theory*, Longman, Harlow, 1996. 2.1.2

[173] Woodruff, A., Landay, J., and Stonebraker, M., "Constant density visualizations of non-uniform distributions of data," *Proceedings of UIST'98*, 1998, pp. 19–28. 1.2.1, 3.1, 4.4, 4.4.1

[174] Woodruff, A., Landay, J., and Stonebraker, M., "Constant information density in zoomable interfaces," *Proceedings of Advanced Visual Interfaces (AVI)*, 1998, pp. 57–65. 1.2.1, 3.1

[175] Woods, D., "Visual momentum: A concept to improve the cognitive coupling of person and computer," *International Journal of Man-Machine Studies*, Vol. 21, 1984, pp. 229–244. 3.3

APPENDIX A

Included CD-ROM Description

This thesis is accompanied by a CD-ROM attached to the inside back-cover. It contains an electronic presentation of this thesis and some supplementary material.

Electronic thesis document: An electronic version of this thesis is contained on the CD-ROM under the filename `thesis.pdf`. It is in Adobe Portable Document Format (PDF), and requires a compatible program for viewing, such as Adobe Acrobat Reader. In addition, if Adobe Acrobat Reader is used, the animated figures may be interactively viewed. This requires Acrobat Reader version 5 or above.

Animated figures: Due to technical limitations, the animated figures are not embedded directly within the `thesis.pdf` file. Rather, each animated figure is stored as an individual file in the `anim` directory on the CD-ROM. Each animated figure is available in two formats, `PDF` and `AVI`. The PDF format figures may be viewed with Adobe Acrobat Reader (and this is what is done when an animated figure is clicked in the main thesis document). The AVI format figures may be viewed using any media player that is able to view movie files in the AVI container format, encoded using the XviD codec. (The AVI format figures are primarily intended as a backup, in case the PDF figures do not properly display or animate.) Table A.1 lists the base filename of each animated figure.

In addition, the accompanying `minepump-tour.wmv` file is contained in the `anim` directory. It may be viewed using any media player that is able to view movie files in the WMV container format, encoded using the Windows Media Video codec, such as Windows Media Player.

Data files: Two types of raw data files are included on the CD-ROM that pertain to the empirical evaluation presented in Chapters 5, 6 and 7 — *input data files*, such as corpus tree and clustered graph files and navigation logfiles, and *output result files*, which are the results of the quality measures after running the experiments. The data files are contained in the `data` directory. This directory is divided into `chapter5`, `chapter6` and `chapter7` directories, each of which is further sub-divided into `input` and `results` directories. The actual data files are then arranged as necessary in these locations.

Adobe Acrobat Reader: Installation files are included in the `acrobat` directory of the CD-ROM for the latest version of Adobe Acrobat Reader for each of the Windows, Macintosh and Linux operating systems.

Animated Figure	Page Num	Base filename
2.38	68	`2x2extra-proper-measures-3d-anim2`
4.8	102	`node-expand-inclusion-animation-anim-reverse`
4.9	103	`node-expand-tipover-animation-anim-reverse`
4.12	104	`hv-rotate-linear-anim`
4.13	105	`hv-rotate-circular-anim`
4.14	105	`hv-rotate-orthogonal-anim`
4.15	107	`hv-rotate-tipover-anim`
4.16	108	`expand-node1-anim`
4.17	108	`expand-node2-anim`
4.19(c)	110	`jewelrybox-anim-standard-anim`
4.21(c)	114	`jewelrybox-anim-nodeorder-anim`
4.22(c)	115	`jewelrybox-anim-stable-anim`
4.26	121	`c488-example-meta-edge-anim-anim-good`
4.27	122	`edge-simple-linear1-anim`
4.29	123	`edge-problem1-anim`
4.31	124	`edge-problem1-improved-anim`
4.34	130	`monotone-se-to-ne-anim`
4.35	130	`monotone-counter-example-2-anim`
4.37	132	`monotone-counter-example-3-anim`
4.39(b)	133	`edge-bend-insert-anim`
4.39(b)	133	`edge-bend-insert-anim-reverse`
4.41	134	`lcs-insert-front1-anim`
4.42(a)	137	`lcs-simple2-anim`
4.42(b)	137	`lcs-simple3-anim`
4.43	138	`lcs-simple4-anim`
4.45	141	`lcs-more-insert1-anim`
4.47	143	`lcs-nontrivial1-anim`
4.50(a)	147	`oel-simple-nonmon-removed-anim-anim`
4.50(b)	147	`oel-simple-nonmon3-removed-anim-anim`
4.52	150	`oel-simple-nonmon-case-1-remove-bad-anim`
4.53	151	`oel-simple-nonmon-case-1-remove-good-anim`
4.55	152	`oel-simple-nonmon-case-2-remove-bad-anim`
4.56	153	`oel-simple-nonmon-case-2-remove-good-anim`
4.58	154	`spiral1-anim`
4.60	156	`edge-problem1-good-anim`
4.61	157	`node-expand-meta-edge-anim`

Table A.1: Reference of the filename for each animated figure.