# Inclusion Tree Layout Convention: An Empirical Investigation

**Kevin Pulo** **Masahiro Takatsuka**

School of Information Technologies
University of Sydney,
NSW, Australia, 2006.
Email: {kev,masa}@it.usyd.edu.au

## Abstract

The inclusion tree layout convention involves drawing trees as nested rectangles rather than the more common node-link diagrams. Finding good inclusion layouts presents some unique challenges, for example, the quantification of what is meant by the "size" of a rectangle. This paper empirically evaluates and investigates several rectangle size measures for their usefulness in the inclusion tree layout convention. We find that the area size measure, commonly used in graph drawing, is very poorly suited to the inclusion layout convention, whilst size measures based on the aspect ratio of the layout are more appropriate and give better results.

*Keywords:* Tree, inclusion, layout, empirical, size measure

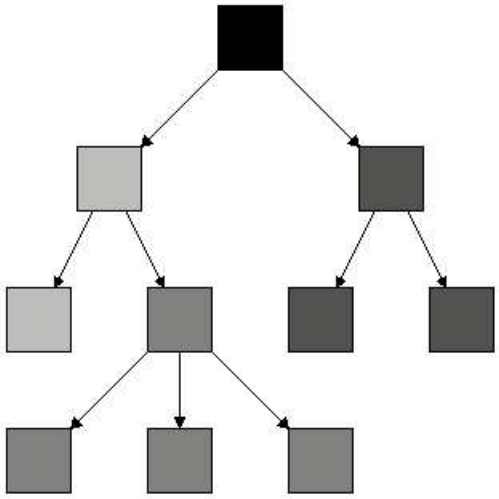
## 1 Introduction

The *inclusion tree layout convention* (Eades, Lin & Lin 1993) is an alternate method of drawing trees where the parent–child relationship is visually represented by the child node being completely contained within the parent node. For simplicity, nodes are usually represented as rectangles. The familiar *classical tree layout convention* draws the tree in a "level" fashion, where the $y$ coordinate of a node is proportional to its depth $k$ from the root, with lines drawn between the child and parent nodes. Figure 1 illustrates an example tree in both the classical and inclusion conventions.

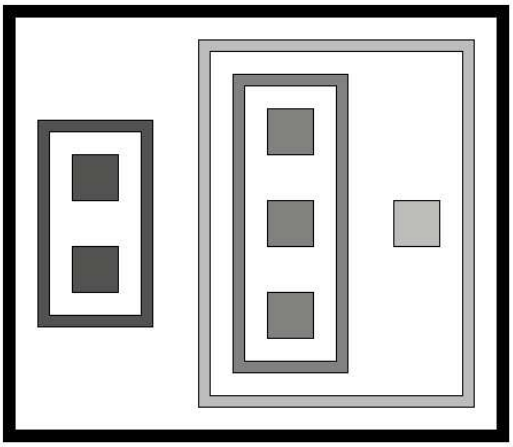
This paper investigates the use of the inclusion tree layout convention in terms of several real-world trees. Several rectangle size measures are compared for their usefulness in efficient inclusion tree layout algorithms. The results show that the area measure, which is commonly used in graph drawing, is inferior to the aspect ratio based measures (such as the minimum enclosing square measure) for the inclusion tree layout convention.

Our investigation is motivated by one of the most fundamental problems in information visualisation: the detail–context tradeoff. In any fixed size display only small amounts of information can be displayed at high detail, resulting in a lack of context (and





(a) Classical Layout Convention



(b) Inclusion layout Convention

Figure 1: An example tree in classical and inclusion layout conventions.

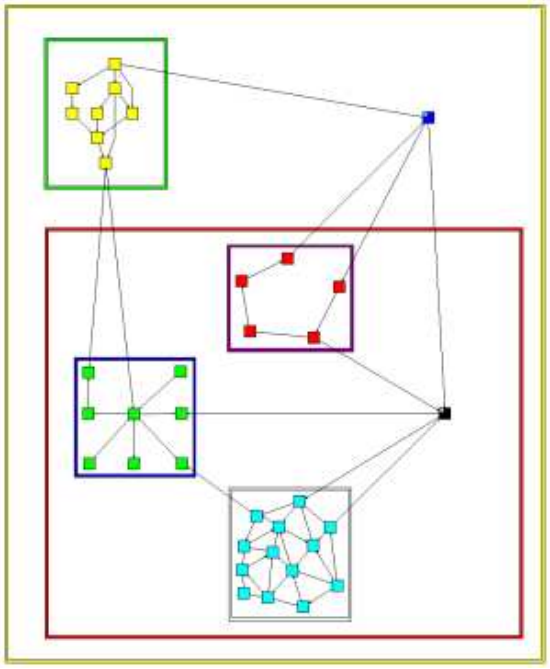


Figure 2: An example of a clustered graph.

vice-versa). This is generally resolved by using *geometric zooming* techniques, such as Focus + Context (for example, fisheye lens, hyperbolic browser) and Overview + Detail, to distort the visualisation into limited regions of high and low detail (Card, Mackinlay & Shneiderman 1999). However, a better solution is *structural zooming*, where the level of data included in the visualisation is varied, rather than simply distorting the full visualisation.

We are interested in applying this technique to graphs, in particular, *clustered graphs*, which support varying levels of detail by defining a recursive clustering of related nodes. Clustered graphs are most often visualised by drawing the contents of each cluster inside a rectangle representing that cluster, as shown in Figure 2. This allows clusters to be "summarised" by simply drawing the cluster rectangle without its contents. Before tackling the problem of structural zooming of clustered graphs, we first study the properties of the simpler inclusion tree layout, which is effectively a clustered graph with no edges.

## 2 Inclusion Tree Layout

### 2.1 Definitions

The formal definition of an inclusion layout for a tree $T$ is a rectangle $R_u$ in the plane $\Re^2$ for each node $u$ of $T$, such that

- if $u$ has a child $w$ then $R_w$ is within $R_u$, and
- if $u$ has children $v$ and $w$ then the rectangles $R_v$ and $R_w$ do not overlap and are separated by a distance of at least $\delta$.

We are interested in inclusion layouts that have a small overall size given sizes of the leaf nodes. This is because, in practice, nodes must contain text and the available screen space is limited.

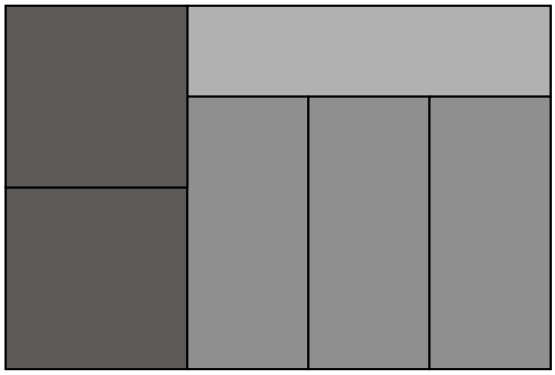


Figure 3: Treemap for the tree shown in Figure 1.

The inclusion tree layout convention is similar to *treemaps* (Johnson & Shneiderman 1991), a space-filling technique for drawing trees in the plane. Figure 3 shows an example treemap of the tree shown in Figure 1.

One disadvantage of the inclusion tree layout is that it does not scale well to very deep trees. It can require exponential area (or exponentially small resolution) in terms of the number of nodes, which means that in a practical sense it is not very useful for trees with depth greater than about 4 or 5. Thus the visualisation of large trees in the inclusion tree layout convention requires a dynamic navigation system, which shows only a small subsection of the entire tree at any given time. The user may then change the view by expanding or collapsing nodes, which may require the inclusion layout to be recomputed or otherwise adjusted. In this paper we examine the properties of static inclusion tree layouts in order to better understand them, and hope to subsequently develop dynamic layout methods.

We consider several measures of the "size" of a rectangle. For a rectangle of width $x$ and height $y$, these are:

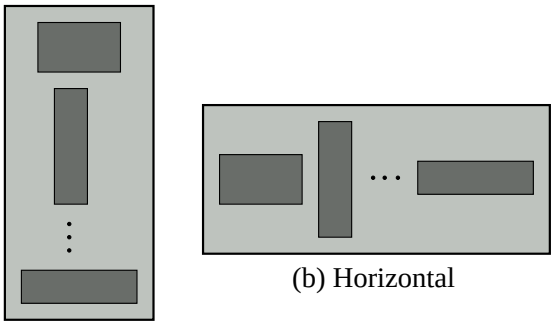
- area: $\psi(x, y) = xy$
- perimeter: $\psi(x, y) = x + y$
- minimum enclosing square: $\psi(x, y) = \max(x, y)$
- square aspect ratio: $\psi(x, y) = \left|\frac{x}{y} - 1\right|$

The only restriction on the size measure is that it must be non-decreasing in both dimensions, that is, $\psi(a, b) \geq \psi(c, d)$ whenever $a \geq c$ and $b \geq d$.

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, most monitors have $r = 4/3$).

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

For simplicity, this paper considers only the square $r = 1$ cases.



(a) Vertical

(b) Horizontal

Figure 4: The two different types of arrangements considered.

The fundamental problem for inclusion layout is as follows:

> **Minimum Inclusion Layout Problem** (MILP): Given a tree $T$ and a width $X_v$ and 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 $X_v \times Y_v$.

If we consider the tree in which every non-root node is a leaf, we can see that MILP is equivalent to a 2 dimensional bin packing problem, and is thus NP-hard (Martello & Vigo 1998). However, a polynomial time dynamic programming solution exists when the problem uses integer coordinates. For binary trees, (Eades et al. 1993) show that MILP can be solved in time $O(Mn)$, where $M$ is the sum of the (integer) widths of the leaf nodes and $n$ is the number of non-leaf nodes. In fact this result can be extended to trees with a maximum degree $d$, as shown in (Hong, Eades & Quigley 2002).

We consider only two possible ways of arranging the children of a node, horizontal and vertical, as shown in Figure 4. This is due in part to the binary tree heritage of the algorithm, but also helps to keep the computational complexity under control by avoiding the full bin packing problem.

### 2.2 Algorithms

The simplest 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 each node is touched exactly once in the traversal, the greedy algorithm has a linear running time. As is often the case with greedy algorithms, it is easy to find counter-examples illustrating how the greedy algorithm may make globally poor decisions based on local conditions.

The dynamic programming algorithm stores a list of possible solutions for each node. Each possible solution is a complete arrangement of the descendants of that node, specified recursively in terms of the possible solutions of the children nodes. The algorithm then 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 of all of the possible overall solutions available, and can then traverse this list to find the best solution according to a given size measure. This solution then specifies a horizontal or vertical arrangement for each non-leaf node, which the algorithm can implement from the leaves upward (since the leaf sizes are fixed) in order to obtain the final solution.

However, such an exhaustive search is clearly not efficient, since the number of possible solutions for a node grows exponentially with the number of descendant nodes. The concept of *dominating rectangles* is used to reduce the number of solutions to a polynomial. We say that a rectangle $(c, d)$ dominates a rectangle $(a, b)$ if and only if $c \geq a$ and $d \geq b$. Since our size measure is non-decreasing, if $(c, d)$ dominates $(a, b)$ then $(c, d)$ will never be included in an optimal layout, and so may be discarded. Figure 5 illustrates this concept diagramatically. Note that we have associated the width of the rectangle with the $x$ value and the height with the $y$ value. We now modify the algorithm to store only non-dominating possible solutions, rather than all possible solutions. Further, we store them in order of increasing $x$; note that the definition of dominating rectangles means that the $x$ coordinates will be unique and the list is also in order of decreasing $y$. For example, in Figure 5 the rectangles A, B, D could be a list of non-dominating possible solution sizes. Now when constructing the list of possible solutions for a node we can simply "merge" the lists of the children in a fashion similar to merge sort. We take a possible solution from each child, add the composition of these possible solutions only if it doesn't dominate the previously added solution, and then advance to the next possible solution for the necessary children. This 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.

### 2.3 Properties

We now examine the properties used empirically investigate the different inclusion layout size measures and algorithms. 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 6 illustrates this concept, while Figure 7 shows the the non-dominating function for the tree shown in Figure 1. Figure 7 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 6, 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 larger of

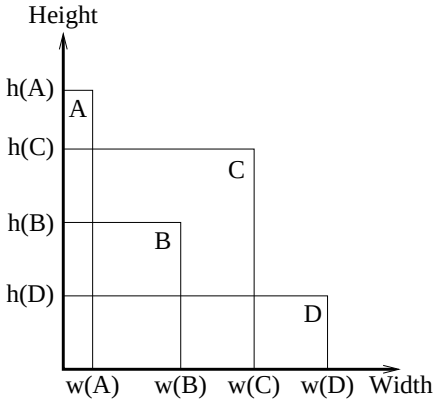



Figure 5: Dominating rectangles. Rectangle C dominates B. Rectangles A, B, D are non-dominating, rectangle C is dominating.

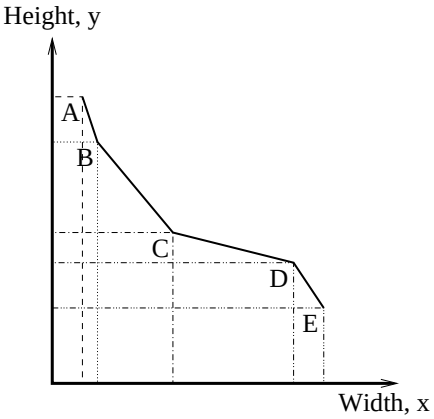



Figure 6: Non-dominating rectangles as a function. Note that as $x$ values increase, the $y$ values decrease (strictly) monotonically.

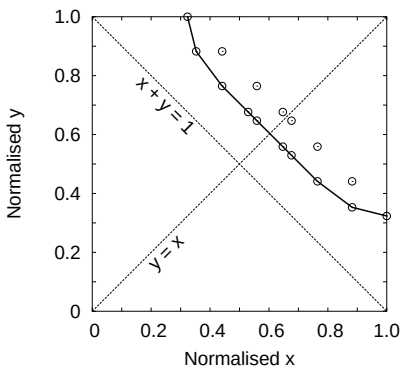



Figure 7: The non-dominating function plot for the tree in Figure 1. The non-dominating function is indicated by the solid line, while possible layout solutions are indicated by circles.

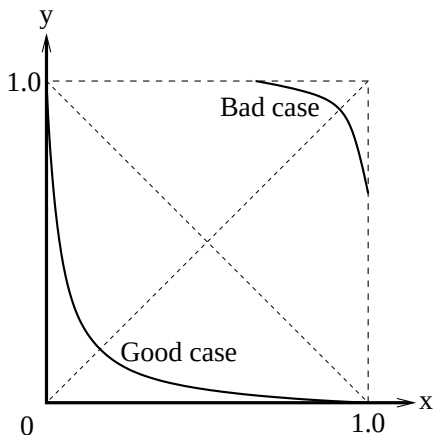



Figure 8: General quality of non-dominating functions.

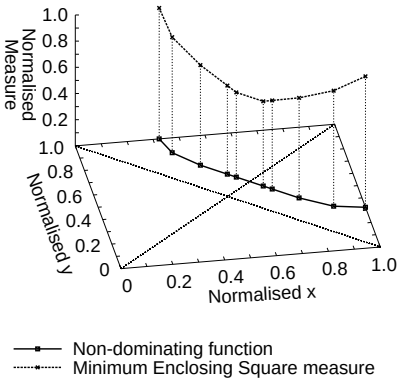



Figure 9: Plotting a size measure above the non-dominating function of Figure 7.

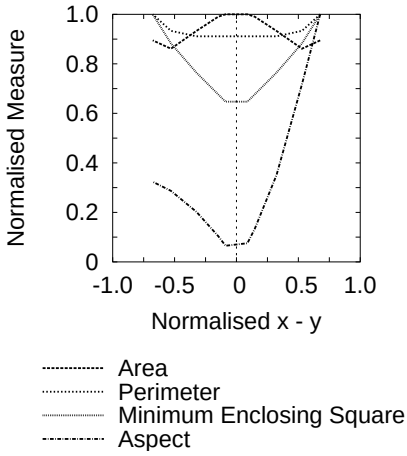



Figure 10: Various size measure plots for the tree in Figure 1.

the height of the tallest non-dominated layout and the width of the widest non-dominated layout is used as the normalisation factor, and the corresponding layout is called the *normalisation layout*.

Figure 8 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 8 are poor candidates for inclusion layout. This is because there is only a small variation between the possible layouts and all have $x$ and $y$ dimensions similar to the maximum $x$ or $y$ dimension, and 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 examine is how the different size measures vary with the different non-dominating solutions. We can use the third dimension to indicate the value of the chosen measure for each layout in the non-dominating function, as shown in Figure 9. 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 we use a two dimensional variant as shown in Figure 10. The $x$ axis of this plot is $x - y$ from Figure 9, and is thus parallel to the line $x + y = 1$ in Figure 7. The vertical dotted line at $x = 0$ corresponds to the line $y = x$ in Figure 7. The dynamic programming algorithm will choose the layout with the minimum value for a given measure.

## 3 Empirical Results

This section presents the results of using the inclusion tree layout convention on various real world trees.

### 3.1 Data

#### Design behaviour trees

*Design behaviour trees* (DBTs) are a new method for representing requirements in software engineering (Dromey 2002). They are constructed by composing the behaviour trees for each functional unit in a software system, and allow the system to effectively be directly built out of its functional requirements, rather than the more traditional activity of building a system which satisfies those requirements. 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 classical and inclusion conventions in Figure 11. One observation about these trees is that they contain many nodes of (out) degree of 1, along with the occasional node of degree 4 or more.

| Tree | Number of Nodes | Number of 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 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.

#### Ontologies

*Ontologies* are formal descriptions of concepts and relationships which exist in a given domain (Gruber 1993). They are commonly used in artificial intelligence 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. We consider two taxonomies:

- The Multi-Sensory Taxonomy (MST) presented in Keith Nesbitt's PhD thesis (Nesbitt 2002), with 290 nodes.
- The Enterprise ontology (Uschold, King, Moralee & Zorgios 1998) from the Ontolingua server at Stanford's Knowledge System Laboratory (Farquhar, Fikes & Rice 1996), with 95 nodes. Figure 12 shows the class hierarchy of this ontology in classical and inclusion conventions.

Compared to DBTs, these trees have less nodes of degree 1 and are generally broader.

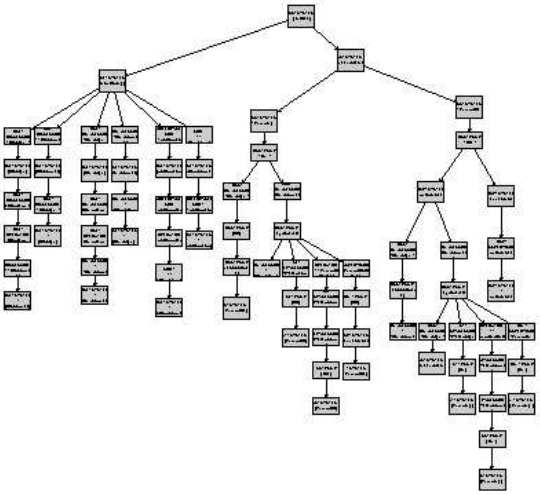
### 3.2 Results

Figure 13(a) shows the non-dominating function plot (as shown in Figure 7) for all of the input trees. Figures 13(b)–13(e) show the results of the four size measures for all of the input trees (as shown in Figure 10). Table 1 lists the input trees along with the number of nodes and the number of non-dominating layouts.

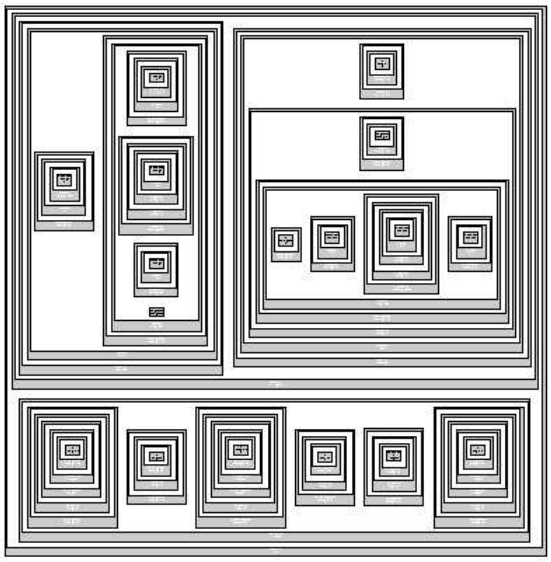
### 3.3 Discussion

#### 3.3.1 Non-dominating functions

In Figure 13(a) we can see that the position and concavity of the non-dominating functions varies considerably between the input trees. Those with the best

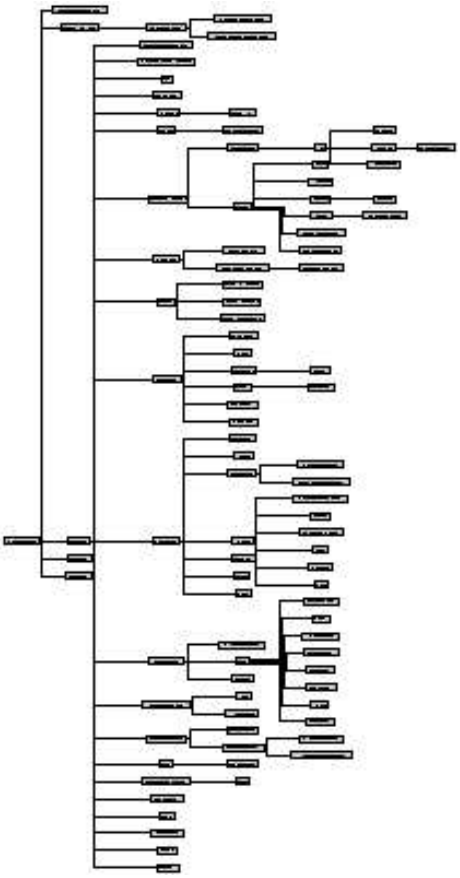


(a) Classical Layout Convention

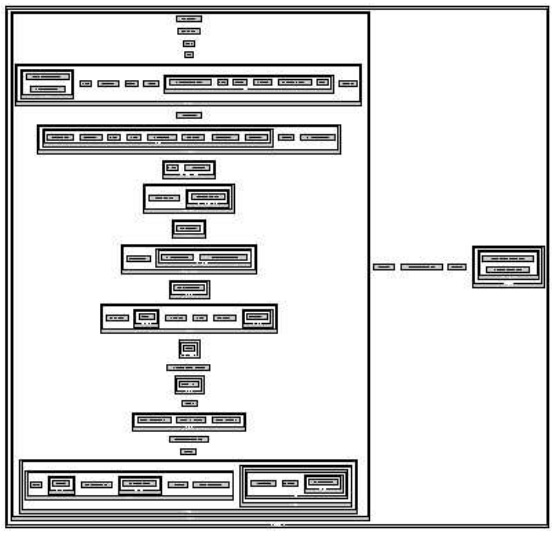


(b) Inclusion Layout Convention (Minimum Enclosing Square)

Figure 11: The Mine Pump DBT



(a) Classical Layout Convention



(b) Inclusion Layout Convention (Minimum Enclosing Square)

Figure 12: The Enterprise-Ontology class heirarchy.

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 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 $\lesssim 60$ 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 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, 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.

### 3.3.2 Measures

Figure 13(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 inclusions layouts. All of the trees exhibit very unstable, jagged plots and are only in the region between approximately 0.5 and 1.0. In fact, some have plots which are clearly an inverted 'U' shape, indicating that for those trees, the layouts with the minimal area have extreme aspect ratios.

Figure 13(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 13(a), and so what this shows is equivalent to Figure 13(a) "rotated" by $45^o$, where the line with $y = 1.0$ and $-1.0 \leq x \leq 0$ in Figure 13(c) corresponds to the line with $y = 1.0$ and $0 \leq x \leq 1.0$ in Figure 13(a), and similarly $y = 1.0$ and $0 \leq x \leq 1.0$ in Figure 13(c) corresponds to $x = 1.0$ and $0 \leq y \leq 1.0$ in Figure 13(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 not surprising, since this plot is effectively equivalent to Figure 13(a).

Figure 13(d) shows a plot of the aspect ratio size measure. In this plot we can see that all of 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 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 13(a), the line where layouts are square.

Figure 13(e) shows a plot of the minimum enclosing square size measure. In this plot we can see that this measure also has a definite minimum at $x = 0$. This is because the minimum enclosing square will be minimised when the layout is itself square, which corresponds to the line $x = 0$ ($y = x$ in Figure 13(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, it will always choose the one with the smaller edge length. For algorithms which take dominating rectangles into account, such as the dynamic programming one, this makes no difference, since 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 don't 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 13(e).

In Table 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 $O(Mn)$, as described in Section 2.1, rather than with the number of nodes.

### 3.3.3 Path compression

As noted in Section 3.1, the DBTs have more nodes of degree 1 than the ontologies. Nodes of degree 1 are particularly poor when drawn with the inclusion layout convention, as they result in a single nested rectangle inside another. For paths of nodes of degree 1, these nested margins add considerable visual complexity and waste screen space, as can be seen in Figure 11.

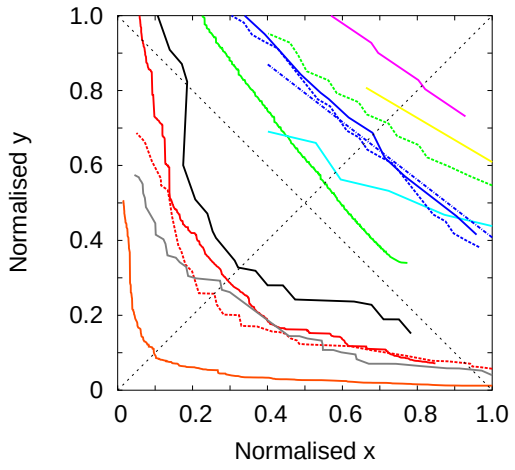
A better solution for the inclusion layout convention in this case is to compress each of these paths of nodes into a single representative node. If desired, the representative node can be scaled by an amount proportional to the length of the compressed path. When this occurs at leaf nodes, the resultant 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 save some screen space. However, when the compressed nodes contain text, an application specific textual summary should be used for the text of the representative node.

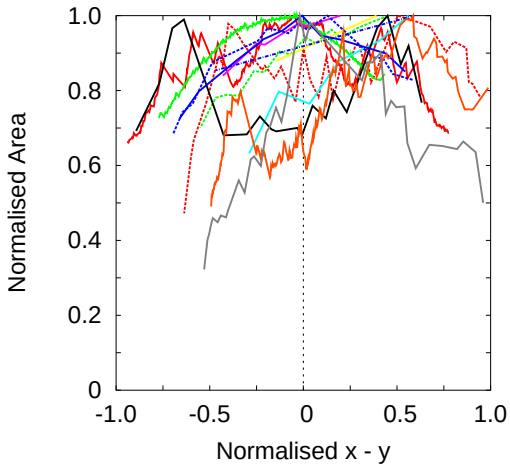
Figure 14 shows the results of applying path compression to the Mine Pump DBT from Figure 11. From Figure 14(a) we can see that the inclusion layout is easier to understand with paths compressed, although a visual cue would be useful to regather some of the lost information. Figure 14(b) shows the same 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 generated for the representative nodes. Figure 14(c) shows that the non-dominating functions of the compressed trees are only marginally worse than the original and have the same basic shape.

## 4 Conclusion

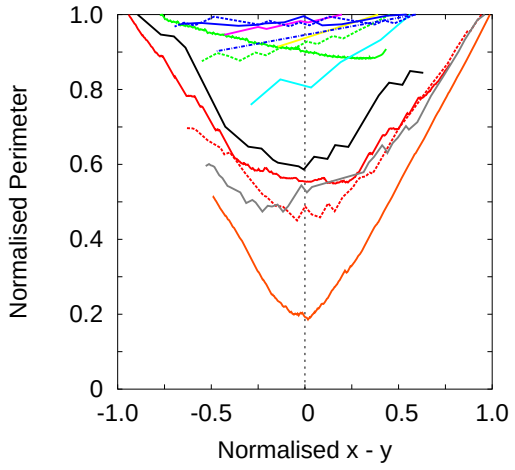
We have presented an empirical investigation of the inclusion tree layout convention, based on design behaviour trees from Software Engineering and ontological class hierarchies from Artificial Intelligence. Several common measures of the size of an inclusion layout have been examined. We find that the area size measure is a poor measure for the inclusion layout convention, and that measures which seek to acheive a desired aspect ratio (such as the minimum enclosing rectangle size measure) are more suitable.



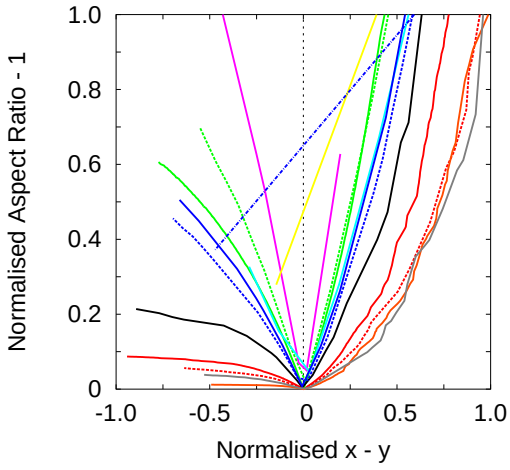

(a) Non-dominating possibles



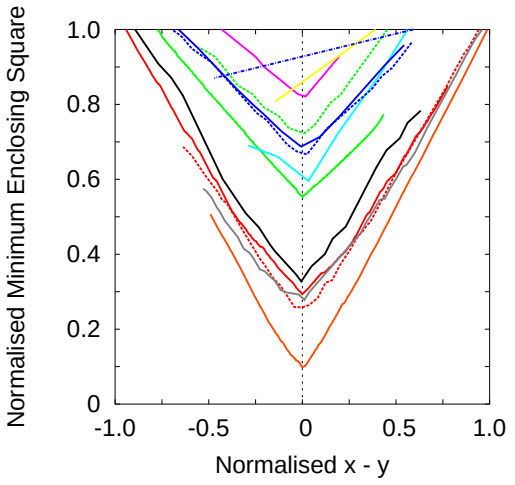

(b) Area measure




(c) Perimeter measure



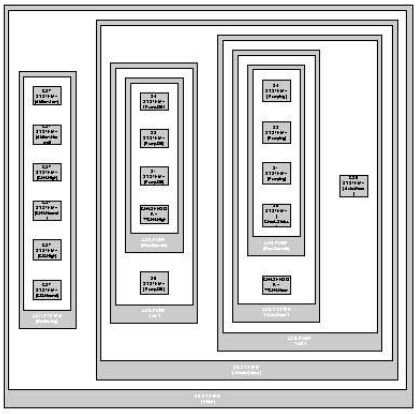

(d) Aspect Ratio measure



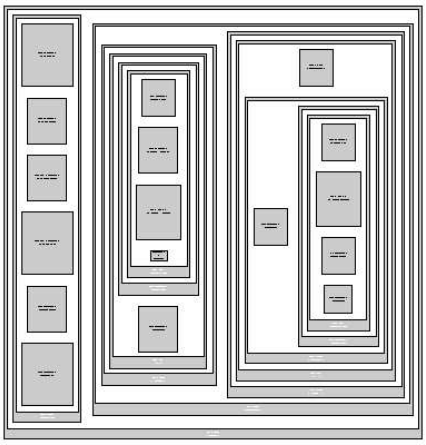

(e) Minimum Enclosing Square measure


(f) Key

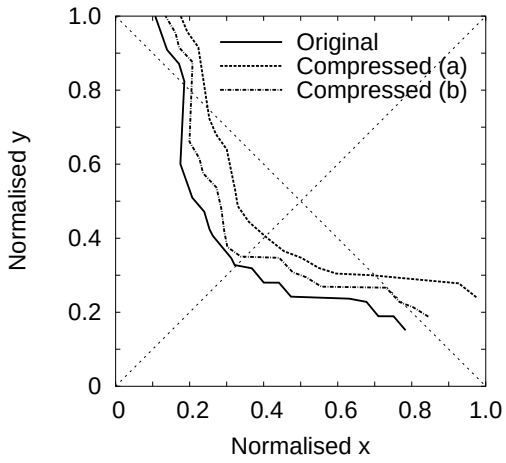
Figure 13: Result plots for empirical data.



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



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




(c) Non-dominating functions

Figure 14: Results of applying path compression to the Mine Pump DBT shown in Figure 11.

## References


Card, S. K., Mackinlay, J. D. & Shneiderman, B (1999), *Readings in Information Visualization*, (Chapter 4), Morgan Kaufmann.

Dromey, R. G. (2002), Genes, Jigsaw Puzzles and Software Engineering *in* 'Proceedings of the 9th Asia Pacific Software Engineering Conference (APSEC) 2002', to appear.

Eades, P., Lin, T. & Lin, X. (1993), 'Two Tree Drawing Conventions', *International Journal of Computational Geometry and Applications* **3**(2), 133–153.

Farquhar, A., Fikes, R., & Rice, J. (1996), The Ontolingua Server: A Tool for Collaborative Ontology Construction. Stanford Knowledge Systems Laboratory, *KSL Technical Report 96-26.* http://ontolingua.stanford.edu/

Gruber, T. R. (1993), 'A translation approach to portable ontologies', *Knowledge Acquisition*, **5**(2), 199–220.

Hong, S. H., Eades, P. & Quigley, A. (2002), 'A Three Dimensional Drawing Algorithm for Series-Parallel Digraphs', to appear.

Johnson, B. & Shneiderman, B. (1991), Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures, *in* 'Proc. IEEE Visualization 1991', San Diego, pp. 284–291.

Martello, S. & Vigo, D. (1998), 'Exact Solution of the Two-Dimensional Finite Bin Packing Problem', *Management Science* **44**(3), 388–399.

Nesbitt, K. (2002), Designing multi-sensory metaphors for interacting with abstract data in virtual environments, Ph.d., University of Sydney, to appear.

Uschold, M., King, M., Moralee, S. & Zorgios, Y. (1998), 'The Enterprise Ontology', *The Knowledge Engineering Review*, Vol. 13, Special Issue on Putting Ontologies to Use (eds. Uschold. M. and Tate. A.).