

Visualizing Argument Structure

Peter Sbarski¹, Tim van Gelder², Kim Marriott¹, Daniel Prager², and Andy Bulka²

¹Clayton School of Information Technology,
Monash University, Clayton, Victoria 3800, Australia,
{Peter.Sbarski, Kim.Marriott}@infotech.monash.edu.au

²Austhink Software,
Level 9, 255 Bourke St, Melbourne, Victoria 3000, Australia,
{tvg, dap, andy}@austhink.com

Abstract. Constructing arguments and understanding them is not easy. Visualization of argument structure has been shown to help understanding and improve critical thinking. We describe a visualization tool for understanding arguments. It utilizes a novel hi-tree based representation of the argument’s structure and provides focus based interaction techniques for visualization. We give efficient algorithms for computing these layouts.

1 Introduction

An *argument* is a structure of claims in inferential or evidential relationships to each other that support and/or refute the main proposition, called the *conclusion* [1, 2]. The ability to create arguments, understand their logical structure and analyse their strengths and weaknesses is a key component of critical thinking. Skill and care is required to correctly identify the underlying propositions of an argument and the relationships between the different propositions, such as rebuttal and support. For this reason diagrammatic representations of arguments, called *argument maps*, have been suggested that more clearly display the evidential relationships between the propositions which make up the argument [3].

Although the earliest modern argument maps can be traced to J.H. Wigmore, who used it for complex evidential structures in legal cases [1], argument mapping is still not common. One reason for this is that revising maps sketched out with a pen on paper isn’t very practical [3]. Since the late nineties, however, specialised software for argument mapping has been developed. A recent study showed that argument mapping helped understanding arguments and enhanced critical thinking. The study also showed that the benefits were greater with computer based argument mapping [3].

However, current computer tools for argument mapping are quite unsophisticated. They provide, at best, poor automatic layout and do not utilize interactive techniques developed for other computer-based information visualization applications. In this paper we present a new computer tool for argument mapping that addresses these deficiencies. Three main technical contributions of the paper are:

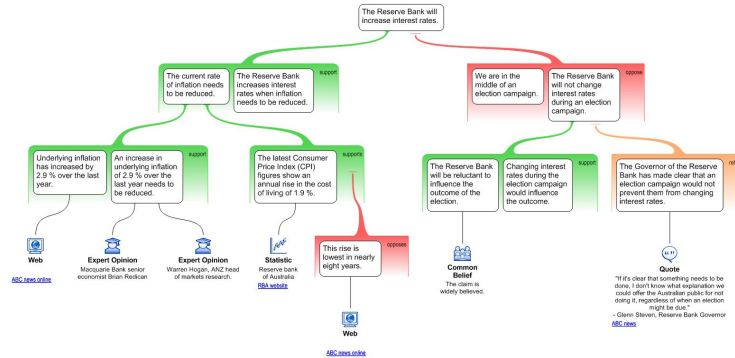


Fig. 1. Example argument map: debating a possible interest rate increase. In fact, in October 2008, the Reserve Bank of Australia did increase interest rates during an election campaign and the incumbent political party lost office.

- A new visual representation for arguments. The key novelty in the representation is to allow propositions on the same level to be grouped into “compound nodes” which denote logical conjunction and which support multi-premise reasoning. An example is shown in Figure 1. (Described in Section 2)
- Real-world arguments are often large and complex. Our tool provides focus-based contextual layouts for visualizing large argument maps. (Described in Section 3)
- The final contribution of the paper are algorithms for laying out argument maps in standard and contextual layout styles. The algorithms are non-trivial extensions to algorithms for laying out standard layered trees. (In Section 4).

Despite its potential usefulness, there has been relatively little work on computer-based visualization of arguments. In this section we critique the five most advanced tools that we are aware of are—*Reason!Able* [1], *Athena Standard* [4], *Compendium* [5], *Araucaria* [6], and *Argunet* [7].

The first difference between the tools is the basic visual representation for the argument structure. *Reason!Able* and *Araucaria* use a layered tree representation which forces the logical structure of the argument to be visible, while *Athena Standard*, *Compendium* and *Argunet* allow more free-form representation of arguments as networks with directed edges. *Rationale* uses a novel layered hi-tree representation which we believe better displays the logical structure of the diagram and handles multi-premise reasoning.

It is fair to say that none of the existing tools provide good automatic layout. This is one of their biggest deficiencies. The two tools *Reason!Able* and *Araucaria* based on layered trees provide fully automatic layout which the user cannot control. *Reason!Able* uses a standard Sugiyama-based layered graph layout algorithm to repeatedly re-layout the graph after user interaction. This lead to narrow layouts but means that the user cannot control the order of a nodes’ children and successive layouts did not necessarily preserve the order of nodes

in each layer, which is quite confusing for the user. Tree layout in *Araucaria* appears to use a non-standard tree layout algorithm which creates quite uncompact layout and does not center nodes above their children. Construction of arguments is difficult and requires the user to first create and load a text file with the text for the argument. It is difficult to re-arrange nodes on the canvas and the order of children in the node depends upon the construction order.

The network based tools—*Athena Standard*, *Compendium* and *Argunet*—behave more like standard graphical editors in which the primary technique for layout is explicit user positioning of elements. *Compendium* and *Argunet* provide some simple placement tools which the user can explicitly invoke to re-layout parts of the diagram. However, these lead to overlapping nodes and generally do not seem well-integrated into the tools.

Sophisticated aesthetically pleasing automatic layout is one of the great strengths of our tool and relies on the algorithms described in this paper. While fully automatic, ordering of children is totally under the user’s control and the user has the ability to control the compactness of the layout.

The other main advantage of our tool over previous argument mapping tools is more powerful techniques for interactive exploration of larger argument maps. Like previous tools it provides an overview+detail view of the map, allows the user to pan+zoom the detail view, and to collapse and expand sub-arguments. But, in addition, it provides focus-based tools for emphasizing the structure of the argument around a focal node.

2 Representing Argument Structure

The first question we must answer is how to represent an argument visually. A fairly natural approach (previously used in the tools *Reason!Able* and *Araucaria*) is to represent argument maps by a tree in which nodes represent propositions, with propositions supporting and refuting their parent propositions. The root of the tree is the conclusion of the argument and the depth in the tree indicates the level of detail in the proof.

A simple tree, however, is not the best structure for representing arguments. One issue is that it does not adequately represent multi-premise arguments. Typically a single proposition does not provide evidence for or against other propositions by itself, rather it does so only in conjunction with other propositions. For instance, if we have that proposition P is supported by P_1 and P_2 and by P_3 and P_4 then representing this as a tree with root P and the P_i ’s as the root’s children loses the information that it is the conjunction of the propositions that support P not the individual propositions. The second issue is that the reason for inferring a proposition from other propositions is—itself—a component of the argument and a well-designed representation should allow propositions supporting and refuting the inference itself. Thus, in our example, the reason “ P_1 and P_2 imply P ” is also part of the argument and may need support.

A straightforward approach is to modify the layered tree representation so as to use a bi-partite tree in which propositions alternate with inference rules.

However, this is not a particularly compact representation and does not visually distinguish between propositions and rules of inference. Instead, we have chosen to use a representation for the bi-partite tree in which the child relationship is represented using containment then links on alternate levels. We call the resulting tree-like structure a *hi-tree* because of its similarity to a hi-graph [8]. We have not found examples of hi-trees before, although for our application they provide an elegant visual representation.

An example of our hi-tree representation is shown in Figure 1. The root of the tree is the conclusion whose truth is to be determined by the argument. Propositions can have reasons supporting them. Usually these reasons are represented by a compound node that contains the propositions which are the basis for the reason, but they may also be a “basic” reason such as a quote or common belief. Notice in the example how words such as “oppose” and “support” indicate the evidential relationship of the reason to its parent proposition. This evidential relationship is also indicated using color: green for support, red for oppose and orange for rebuttals. The author may also indicate how strongly they believe a particular reason: this is shown visually by the thickness of the link between the reason and its parent proposition. Also note how the compound node represents the inference rule and can itself have supporting or refuting arguments as shown in Figure 1 where there is an argument opposing the inference that the high current CPI rate implies that the current inflation rate needs to be reduced.

In our tool the *standard layout* for a hi-tree argument map is as a kind of layered tree. This seems like a natural and intuitive visual representation for argument maps that clearly displays their hierarchical structure since the main proposition is readily apparent at the top of the screen, with the layer indicating the depth of the argument. We considered the use of other layout conventions for trees including h-v trees, cone trees and radial trees but felt that none of these showed the structure of the argument as clearly as a layered tree representation.

The *standard layered drawing convention* for ordered trees dictates that [9, 10]:

- LT1 the y-coordinate of each node corresponds to its level,
- LT2 nodes on the same level are separated by a minimum gap,
- LT3 each node is horizontally centered between its children,
- LT4 the drawing is symmetrical with respect to reflection,
- LT5 a subtree is drawn the same way regardless of its position in the tree,
- LT6 the edges do not cross, and
- LT7 the trees are drawn compactly.

We now discuss how we have modified this drawing convention to hi-trees.

To make the discussion more precise we must formalize what a hi-tree argument map is. Like any ordered tree it consists of *nodes*, $N = \{1, \dots, n\}$, and a function $c : N \rightarrow N^*$ which maps each node to the ordered sequence of nodes which are its children. No node, i , can be the child of more than one node: the node of which it is a child is called its *parent* and denoted by $p(i)$. There is one distinguished node, the root node r , which has no parent.

It is a bipartite tree: Nodes are partitioned into *proposition nodes* N_P representing a single proposition, and *compound nodes* N_C which represent reasons. The root is required to be a proposition node. The children of a proposition node must be compound nodes and the children of a compound node must be proposition nodes. Every compound node has a single distinguished child that represents the proposition that the inference used in the reason is valid. This is the last child of the node. We say that the children of a compound node are its *components*. A basic reason, such as a quote, is modelled by a compound node which has one child with no children.

While the underlying bi-partite tree of the argument map is important, so is the *visual tree* which models the visual representation of the hi-tree and captures the requirement that compound nodes and their components occur on the same visual level. If i is a node its *visual children*, $vc(i)$, are, if i is a compound node, the children of its components and if i is a proposition its visual children are simply its children. The *visual parent* of a node i , $vp(i)$, is simply the compound node of which i is the visual child. We define the *visual level* $vl(i)$ of node i inductively as follows:

- If $i = r$ then $vl(r)$ is 1;
- If $p(i) \in N_P$ then $vl(i) = 1 + vl(p(i))$;
- If $p(i) \in N_C$ then $vl(i) = vl(p(i))$

In the layered drawing convention for hi-trees requirement LT1 is modified to the requirement that the y -coordinate of each node corresponds to its visual level.

It is natural to add a requirement to the hi-tree drawing convention that:

HT7 Each compound node is drawn as compactly as possible with its component nodes separated only by a minimum gap.

However, requirement LT3 that each node is placed midway between its children conflicts with this requirement. The problem is that forcing component nodes to be placed next to each other means that if each component is placed centrally between its children then the sub-trees may overlap. The problem and two possible solutions are shown in Figure 2. One solution would be to weaken requirement HT7 and allow components of a compound node to be separated by more than a minimum gap. Another possible solution is to weaken requirement LT3 and only require that component nodes are placed as close as possible to the midpoint of their children. This is the solution we have chosen.

Thus, we define the *standard layered drawing convention* for hi-trees to be:

HT1 the y -coordinate of each node corresponds to its visual level,

HT2 compound nodes on the same visual level are separated by a minimum gap,¹

HT3 the root is midway between its children and each compound node is placed so that each of its components is, as far as possible, midway between its children,

HT4 the drawing is symmetrical with respect to reflection,

HT5 the edges do not cross,

¹ Which in our tool is inversely proportional to how closely the nodes are related.

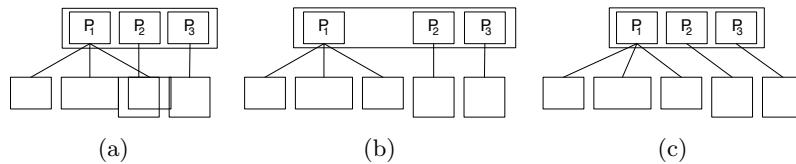


Fig. 2. Requirement LT3 that each node is placed midway between its children may conflict with requirement HT7 that components of a compound node be placed tightly together. As shown in (a), the conflict between requirements LT3 and HT7 can lead to overlapping nodes. One solution, shown in (b), is to weaken requirement HT7. However this leads to larger compound nodes which may exaggerate their importance and can lead to clumping of components. In the example clumping suggests that P_2 and P_3 are somehow more connected or similar to each other than to P_1 which is misleading. Another solution, shown in (c), is to weaken requirement LT3—this is the solution we have chosen.

HT6 the trees are drawn compactly, and
 HT7 compound nodes are drawn as compactly as possible with component nodes separated only by a minimum gap.
 The argument map in Figure 1 illustrates this drawing convention.

3 Handling Large Argument Maps

Real-world argument maps can be quite large with up to a hundred nodes. While this may not sound very large, typical argument maps of this size will not fit on to a single screen or print on a single page using legible fonts because of the large amount of text in the nodes. Thus, a key requirement for any argument mapping tool are techniques to handle larger maps.

Interactive techniques have been extensively studied for visualization of large networks and hierarchical structures and proven very effective, e.g. [11–13]. It is therefore natural to consider their use in argument mapping.

Our tool provides two standard visualization techniques to allow the user to focus on the parts of the map that are of most interest. First, the argument mapping tool provides an overview and detailed view of the argument map. The detailed view can be zoomed and panned. Second, the tool allows the user to control the level of detail in the argument map by collapsing/expanding the sub-argument under a compound or proposition node. These techniques have also been used in some earlier argument mapping tools.

More interestingly, the tool provides a novel focus-node based visualization for argument maps. It allows the user to select a node and choose a “contextual layout” tool which modifies the layout so as to better show the argument structure around that node. Contextual layout tools modify the drawing convention to require that a context of nodes around the focus node are laid out as nicely as possible so as to emphasize their relationship with the focus node, and nodes not in the context are moved slightly away from the contextual nodes and

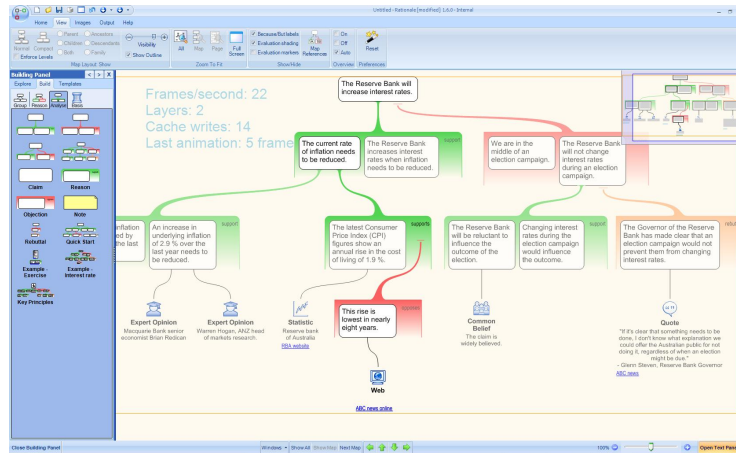


Fig. 3. Screenshot of the tool displaying an example of “ancestors” contextual layout applied to the *Interest Rate* argument map from Figure 1.

optionally faded. Focus nodes can be propositional nodes or compound nodes. While contextual layout bears some similarities to focus based navigation of large networks [13] the details appear to be novel.

Contextual layout has a number of variations which differ in the choice of context: “show ancestors”, “show parent”, “show family” and so on. As an example, in “show ancestors” layout, when a node is selected its parent and other ancestors form the context nodes. They are re-positioned directly above it thus showing all ancestors in a direct line. All other nodes are pushed away from this particular “path”. This type of layout helps to understand the flow of logical reasoning from the selected premise or reason to the final claim, see Figure 3. Another example is “show family”. In this the context comprises all nodes immediately related to the focus node, i.e, parent, siblings and children. The layout centers the parent directly above the focus node, siblings are brought in as close as possible to the focus node and the focus node’s children are grouped directly beneath it.

4 Layout Algorithms

In this section we detail the algorithms we have used to provide the standard layered hi-tree layout and various contextual layouts discussed in the last section.

4.1 Layered Tree Layout

Our layout algorithms are based on those developed for drawing trees using the standard layered drawing convention. Wetherell and Shannon gave the first linear time algorithm for drawing of layered binary trees [14] and Reingold and

Tilford improved this to meet the standard layered drawing convention while still maintaining linear complexity [15].

The Reingold-Tilford algorithm uses a divide and conquer strategy to lay out binary trees. It recursively lays out the subtrees bottom-up and then places them together with a minimum separation between them. The algorithm keeps for each subtree the *left contour*, i.e. a linked list of the leftmost node for each level in the subtree, and the *right contour* of the tree. These are traversed rather than the whole subtree to determine how close the subtrees can be placed together. Walker [16] gave a modification of the Reingold-Tilford Algorithm for n -ary tree layout but has quadratic worst case complexity. Buchheim, Jünger and Leipert [17] improved Walker's algorithm to have linear time. In practice, however Walker's algorithm is very fast and it and the Reingold-Tilford Algorithm are widely used for tree layout.

4.2 Standard Hi-Tree Layout

It is possible to modify the Walker Algorithm (and the other tree layout algorithms) to handle hi-tree layout using the layered drawing convention. The basic algorithm remains the same: recursively lay out the sub-trees bottom-up and then place them together with a minimum separation between the children. Again for efficiency, subtree frontiers and relative positioning of children to their parent and predecessor in the frontier is used.

The key question is how a compound node u is positioned relative to its visual children. Assume u has component nodes v_1, \dots, v_k . By the time u is reached the children of each v_i have been laid out and their frontier computed. There are three steps in computing the layout for the tree rooted at u .

1. For each v_i , the tree T_i with root v_i is computed using the standard Walker Algorithm: the children of v_i are placed as close together as possible (spacing out the enclosed sub-trees where necessary) and v_i is placed on top of the tree midway between its children.
2. Now the T_i s are placed together as closely as possible so as to compute the minimum gap g_i required between v_i and v_{i+1} to avoid T_i and T_{i+1} overlapping
3. We arbitrarily place node u at the x -position $u^x = 0$. This also fixes the position c_i for the component in u corresponding to v_i . We wish to find the position v_i^x for each v_i which minimizes $\sum_{i=1}^k w_i (c_i - v_i^x)^2$ subject to ensuring that for all i , $v_i^x + g_i \leq v_{i+1}^x$ where the weighting factor w_i for each component is some fixed value. For standard layout we have found that setting w_i to the width of the top layer of T_i works well. We use the procedure *optimal_layout* given in [18] to solve this simple constrained optimization problem.

The complexity of the hi-tree layout algorithm remains the same as the complexity of the original Walker Algorithm. The only possible source of additional complexity is the call to *optimal_layout* with the k children of node u . The algorithm has linear complexity in the number of variables passed to it. Since a variable is only passed once, the overall complexity of calling *optimal_layout* is linear.

4.3 Contextual Layouts

Handling contextual layout appears more difficult. However, a nice property of the layout algorithm is that it is parametric in the minimum gap allowed between adjacent nodes and in the weight used to enforce placement of a compound node component near the corresponding sub-tree. By appropriately choosing these it is straightforward to extend them to handle contextual layout.

4.4 Algorithm Evaluation

To evaluate the efficiency of our algorithm we laid out 40 random hi-trees, with 50 to 2000 nodes. We measured layout time for standard layout. See figure 4 for the results. Our results verify our experience with our tool that all of the layout algorithms are more than fast enough to handle layout of even very large argument maps with 1000 nodes in less than 1 second.

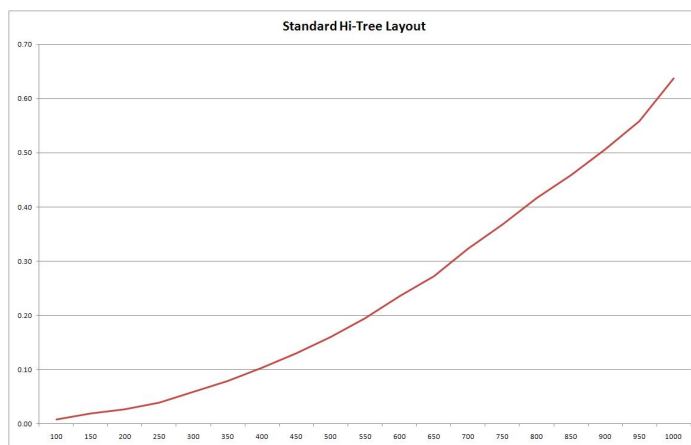


Fig. 4. Performance results. The horizontal axis gives the number of nodes and the vertical axis gives the timing of the layout in seconds.

All experiments were run on a 1.83 GHz Intel Centrino with 1GB of RAM. The algorithms were implemented using Microsoft Visual C# Compiler version 8.00.50727.42 for Microsoft .NET Framework version 2.0.50727 under the Windows XP Service Pack 2 operating system. We used the original Walker algorithm rather than the linear version: as the results show performance of the original algorithm is very fast.

5 Conclusion

We have described a new application of information visualization argument mapping. Critical analysis of arguments is a vital skill in many professions and a necessity in an increasingly complex world. Despite its importance, there has been

comparatively little research into visualization tools to help with understanding of complex arguments. We have described and motivated the design of a particular representation of the argument structure based on a new kind of diagram we call a hi-tree. We have then described basic interactions for visualizing argument maps, such as showing context. Finally, we have given novel algorithms for standard and contextual layout of hi-trees. The algorithms and techniques described here have been extensively tested and evaluated. They underpin a commercial argument mapping tool called “Rationale”². This program supports authoring and visualization of hi-tree based argument maps as well as a simpler standard tree based representation. The illustrations in this paper are from this tool.

References

1. van Gelder, T.J.: Argument mapping with reason!able. *The American Philosophical Association Newsletter on Philosophy and Computers* (2002) 85–90
2. van Gelder, T.J.: *Visualizing Argumentation: software tools for collaborative and educational sense-making*. Springer-Verlag (2002)
3. Twardy, C.: Argument maps improve critical thinking. *Teaching Philosophy* (2004)
4. Athena: <http://www.athenasoft.org/> (2002)
5. Compendium: compendium.open.ac.uk/institute/ (2007)
6. Araucaria: <http://araucaria.computing.dundee.ac.uk/> (2006)
7. Argunet: <http://www.argunet.org> (2008)
8. Harel, D.: On visual formalisms. In Glasgow, J., Narayanan, N.H., Chandrasekaran, B., eds.: *Diagrammatic Reasoning*. The MIT Press, Cambridge, Massachusetts (1995) 235–271
9. Brüggermann-Klein, A., Wood, D.: Drawing trees nicely with tex. *Electronic Publishing* **2**(2) (1989) 101–115
10. Kennedy, A.J.: Drawing trees. *Functional Programming* **6**(3) (1996) 527–534
11. Plaisant, C., Grosjean, J., Bederson, B.B.: Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In: *Proceedings of the IEEE Symposium on Information Visualization, Washington, DC, USA, IEEE Computer Society* (2002) 57
12. Kumar, H., Plaisant, C., Shneiderman, B.: Browsing hierarchical data with multi-level dynamic queries and pruning. Technical Report UMCP-CSD CS-TR-3474, College Park, Maryland 20742, U.S.A. (1995)
13. Huang, M.L., Eades, P., Cohen, R.F.: Webofdav navigating and visualizing the web on-line with animated context swapping. *Comput. Netw. ISDN Syst.* **30**(1-7) (1998) 638–642
14. Wetherell, C., Shannon, A.: Tidy drawings of trees. *IEEE Transactions on Software Engineering* **5**(5) (1979) 514–520
15. Reingold, E.M., Tilford, J.S.: Tidier drawings of trees. *IEEE Transactions on Software Engineering* **7**(2) (1981) 223–228
16. Walker, J.Q.: A node-positioning algorithm for general trees. *Software Practice and Experience* **20**(7) (1990) 685–705
17. Christoph Buchheim, M.J., Leipert, S.: Improving Walker’s algorithm to run in linear time. In: *Graph Drawing*. (2002) 344–353
18. Marriott, K., Moulder, P., Hope, L., Twardy, C.: Layout of Bayesian networks. In: *Australasian Computer Science Conference. Volume 38*. (2005)

² <http://rationale.austhink.com/>