DEXiTree:
A PROGRAM FOR PRETTY DRAWING OF TREES

Marko Bohanec
Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel: +386 1 4773309; fax: +386 1 4773315
e-mail: marko.bohanec@ijs.si

ABSTRACT

This paper presents DEXiTree, a computer program for drawing trees. In principle, DEXiTree is aimed at making nice drawings of attribute trees made by DEXi, a computer program for qualitative multi-attribute decision modelling. Apart from that, DEXiTree is quite a general and powerful tree-drawing program that implements four different tree-drawing algorithms (called *Distribute*, *Align*, *Walker*, and *QP*), draws trees in four different directions (top-down, left-right, bottom-up and right-left) and provides an extensive set of parameters for controlling the appearance of trees and their components. DEXiTree's functionality includes loading a decision model from a DEXi file, interactively designing the tree layout, saving and loading the layout using an XML format, and rendering the drawing in two graphic formats: vector and raster.

1 INTRODUCTION

Trees are a common and very important data structure used in computer science. Trees are used to represent hierarchies such as family trees, organization charts, search trees, taxonomies and file hierarchies. Trees are also heavily used in decision support and decision analysis to represent the structure of models such as decision trees and multi-attribute models.

One such a decision-support computer program is DEXi (Bohanec, 2007). DEXi is aimed at the development of qualitative multi-attribute decision models, which are used in complex real-life decision problems to evaluate and analyse decision alternatives. A multi-attribute model is a hierarchical structure that represents the decomposition of decision problem into subproblems, which are smaller, less complex and possibly easier to solve than the complete problem. In practice, it is essential that such trees are properly visualised so that they could be reviewed by and communicated between DEXi users, and ultimately included in various reports, presentations and publications.

This paper presents DEXiTree, a computer program for drawing DEXi trees. The development of DEXiTree has been directly motivated by DEXi, in particular by its current inability to make nice drawings of multi-attribute models. However, the tree-drawing algorithms implemented in DEXiTree are quite general and could be used for drawing other than just DEXi trees. DEXiTree provides a rich set of parameters for an interactive design of the visual appearance of trees and their components: nodes, arcs and text boxes. DEXiTree is implemented in Borland Delphi and runs under Microsoft Windows. It is publicly available and can be downloaded from http://www-ai.ijs.si/MarkoBohanec/dexitree.html.

This paper is structured as follows. Section 2 describes the four tree-drawing algorithms implemented in DEXiTree. Section 3 presents the functionality of DEXiTree, and section 4 concludes the paper.

2 TREE-DRAWING ALGORITHMS

Tree-drawing algorithms have been extensively studied in the context of graph drawing (Di Battista, et al., 1994; 1999). Although trees have a much simpler structure than general graphs, their drawing – and particularly, "nice" drawing – is far from trivial. The tree layout problem is formulated as follows (Kennedy, 1996): given a labelled tree, assign to each node a position on the page to give an aesthetically pleasing rendering of the tree. We assume that nodes at the same depth are positioned on the same line on the page, so the problem reduces to finding a position horizontally for each node.

A common problem with this setup is that it usually requires a lot of width on the page. The challenge is thus to use the width as effectively as possible, that is, to make trees as narrow as possible. However, this should be combined with the requirement for an "aesthetically pleasing" drawing, which is usually defined by a set of *aesthetic rules* that constrain the node positions in a number of ways:

1. Two nodes at the same level should be placed at least a given distance apart.
2. A parent should be centred over its descendants (lower-level nodes, either immediate descendants or terminal nodes).
3. Drawings should be symmetrical with respect to reflection.
4. Identical subtrees should be rendered identically—their position in the larger tree should not affect their appearance.

The most common approach to the layout problem is the following (Kennedy, 1996). First, draw all the subtrees of a node in such a way that none of the rules are broken. Fit

these together without changing their shape (otherwise rule 4 is broken), and in such a way that rules 1 and 3 are satisfied. Finally, centre their parent above them (rule 2). Clearly, this is a recursive algorithm that gradually positions nodes and subtrees in the bottom-up direction.
DEXiTree implements four tree-drawing algorithms, three of which are based on the above schema. The fourth algorithm takes a different approach based on constrained optimisation.

2.1 Algorithm *Distribute*

We call *Distribute* the algorithm that is probably the simplest "meaningful" algorithm that obeys all the aesthetic rules. The algorithm positions all terminal nodes horizontally in their left-to-right order, allocating to each terminal node its natural width and separating adjacent terminal nodes with the required separation distance. This positioning does not take into account the actual level of terminal nodes in the tree. After positioning the terminal nodes, their parents are recursively centred on the levels above.

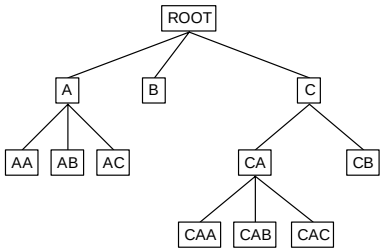

Figure 1: A tree drawn by the *Distribute* algorithm.

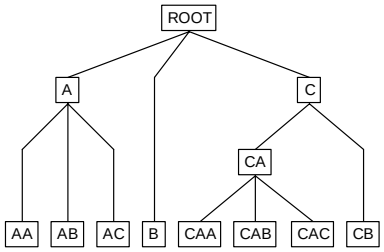

Figure 2: The same tree drawn by *Align*.

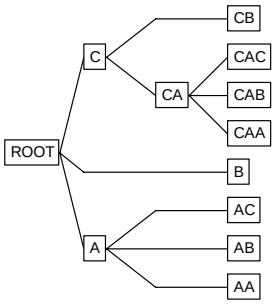

Figure 3: A left-to-right orientation drawn by *Align*.

Figure 1 reveals an obvious disadvantage of *Distribute*: ineffective use of horizontal space. There are too large gaps between the subtrees A and B, B and C, and CA and CB. All these subtrees could have been brought together. This issue is addressed by the algorithm *Walker* (section 2.3). Nevertheless, *Distribute* provides a good basic positioning for a useful variation of the algorithm, called *Align*.

2.2 Algorithm *Align*

The *Align* algorithm is essentially the same as *Distribute*, except that all terminal nodes are brought to the same level at the bottom of the tree (Figure 2). Although this rendition violates rule 4, it has a practical value. Namely, in DEXi, terminal nodes represent input attributes of a multi-attribute model, so it makes sense that they are grouped together and shown at a same level. Also, *Align* is great for drawing trees that are oriented from left (root) to right (terminal nodes), or vice versa. In this case, each terminal node occupies one "line" in the drawing, producing a nice and highly readable layout (Figure 3).

2.3 Algorithm *Walker*

The third algorithm has been originally proposed by Walker (1990). Basically, it is similar to *Distribute*, but it additionally takes into account two important issues:

- Better use of horizontal space, which is achieved by moving subtrees closer together wherever possible (see the subtrees CA and CB, and A and C in Figure 4).
- Handling "orphans", that is, single nodes or small subtrees surrounded by large subtrees and thus having a lot of free space around them (B in Figure 4). These are centred or distributed so as to satisfy rule 3.

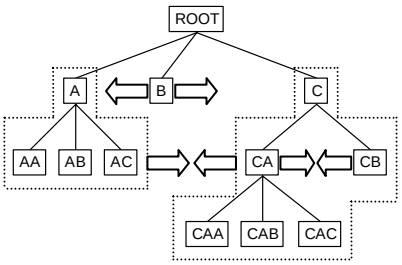

Figure 4: Tree-adjustment operations of *Walker*.

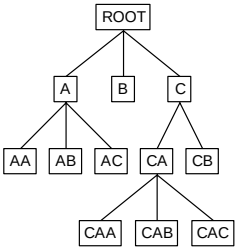

Figure 5: The tree drawn by *Walker*.

The result of these operations performed on our sample tree is shown in Figure 5. Now, the tree is narrower and uses the horizontal space very efficiently.
Walker is one of the best general-purpose tree-drawing algorithms. Its drawings satisfy all the four aesthetic rules. The algorithm is also very efficient: an improved version of

the algorithm, proposed by Buchheim, et al. (2002), runs in linear time with respect to the number of nodes.

2.4 Algorithm *QP*

The fourth algorithm takes a different approach and draws trees simulating a physical system composed of wires, pearls and springs. Imagine that nodes are pearls (of appropriate width), sliding on horizontal wires, which represent tree levels. Suppose that pearls are equipped with appropriate "bumpers" so that they always stay sufficiently apart. Finally, let parents and their children be connected by springs. The idea is to construct such a tool, release the pearls ... and see what happens. Eventually, the system will position itself into some, hopefully nice, tree structure.

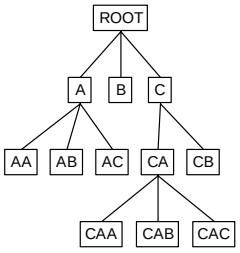

Figure 6: The tree drawn by the *QP* algorithm.

Recently, such approach has been studied for drawing general graphs (Dwyer, et al., 2006). Here, we present our own problem formulation for trees.

Let x_a and y_a denote horizontal and vertical coordinates, respectively, of the centre of some node a. Let the node p be a parent of c. Then, p and c are connected by a spring, whose elastic potential energy E_{pc} is proportional to the squared distance between p and c:

$$E_{pc} \propto d_{pc}^2 = (x_p - x_c)^2 + (y_p - y_c)^2$$

The system will self-organise itself so that the total elastic potential energy E of all springs will be minimal. We assume that the distances between two adjacent tree levels (y_p–y_c) are all equal and constant, so we may discard them from the minimization. The goal is then to minimise the total $E = \Sigma E_{pc}$, which is proportional to

$$\sum_{\forall p,c:\, p \text{ is parent of } c} (x_p - x_c)^2$$

This minimization is subject to constraints: two adjacent pearls placed on a single wire must be separated with at least the node separation distance s. Thus, each pair of adjacent nodes a and b, where a is positioned to the left of b, should satisfy the condition

$$x_b \geq x_a + \tfrac{1}{2}w_a + s + \tfrac{1}{2}w_b$$

where w_a and w_b denote the widths of the respective nodes.

Finally, to guarantee a unique solution, the x coordinate of the root of tree should be set to some constant, typically 0.

In this way, the tree-positioning problem is formulated as a constrained optimisation problem. More precisely, because the objective function is quadratic, this is a quadratic programming (QP) problem (hence the name of the algorithm). In DEXiTree, we use a QP solver based on the Goldfarb-Idnani method (Goldfarb, Idnani, 1983) and adapted from a publicly available Fortran source code of P. Spelluci.

Interestingly, the *QP* algorithm produces very nice and very compact trees, which look "natural" and balanced, even though they generally violate aesthetic rules 2 and 4 (Figure 6). Breaking these rules typically allows *QP* to use horizontal space even better than *Walker*.

3 DEXiTree FUNCTIONALITY

DEXiTree is an interactive Windows program facilitating:

1. Loading a DEXi model from a DEXi (.dxi) file.
2. Interactive design of the drawing (Figure 7) by:
   - choosing between the four algorithms,
   - selecting the drawing direction (section 3.1),
   - modifying drawing parameters (section 3.2).
3. Save the drawing to a file, or copy the drawing to clipboard for transferring it to other programs.

Drawings are rendered in two graphic formats:

- Windows Enhanced Metafile (.emf), which is a vector graphic format, and
- Windows Bitmap (.bmp), a raster graphic format.

In addition, DEXiTree uses its own XML-based ".dxt" file format for storing the current tree structure and drawing parameters. DEXiTree can both load and save these files.

3.1 Tree-Drawing Directions

DEXiTree can draw trees in four different directions: Top-Down, Left-Right, Bottom-Up and Right-Left. The Top-Down direction makes a usual placement so that the root of the tree is shown at the top and subtrees branch downwards. The other three directions respectively correspond to 90-degree counter-clockwise rotations of the tree structure. Displayed text, however, is never rotated.

3.2 Tree-Drawing Parameters

On the right-hand side of its main window (Figure 7), DEXiTree offers an extensive set of parameters for controlling the appearance of trees and their graphical components. In most cases, DEXiTree immediately responds to a parameter change and redisplays the current tree. Parameters are conveniently grouped on two pages, **Tree** and **Node** (Figure 7).

The **Tree** page contains parameters that affect the placement of the tree as a whole:

- horizontal and vertical stretching of the drawing,
- picture borders,
- separation of two adjacent nodes, levels and node boxes,
- parent alignment,
- tree mirroring, and
- background colour of the whole drawing.

The **Node** page controls the display of tree nodes. The user can set drawing parameters individually for each node or

collectively for a group of nodes: all nodes, terminal nodes, or internal nodes in the tree.

Node has two subpages, **Graphic** and **Text**. The former defines graphical properties for displaying tree nodes, such as the minimum and maximum dimensions of the node, its shape, colour, line and fill, vertical alignment of nodes, and positioning of the incoming and outgoing arcs. The **Text** page controls the display of text within nodes: internal text borders, text wrapping, clipping and trimming, line spacing for multi-line text, text positioning within a node, and specifying text font.

6 CONCLUSION

DEXiTree is a small, convenient and publicly available software tool that provides four state-of-the-art tree-drawing algorithms. Basically, it is aimed at drawing DEXi multi-attribute trees, but can also be used to draw other types of trees. DEXiTree is highly interactive and responds immediately to parameter changes. Parameter defaults have been designed so that it is easy to produce a useful drawing without too much effort, but the extensive set of parameters allows an advanced user to control almost any aspect of the graphic output. We believe that DEXiTree has also an educational value for the study of tree-drawing algorithms.

Currently, DEXiTree does not offer any tree-editing capabilities and leaves this task to DEXi. In the future, this might be alleviated by an integration of DEXi and DEXiTree. Also, it is likely that DEXiTree will be extended with some other types of tree-drawing layouts, such as radial, indented or cascade.

References

Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Algorithms for Drawing Graphs: an Annotated Bibliography. *Computational Geometry: Theory and Applications* 4, 235–282, 1994.

Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999.

Bohanec, M.: *DEXi: Program for Multi-Attribute Decision Making, User's Manual, Version 2.00*. IJS Report DP-9596, Jožef Stefan Institute, Ljubljana, 2007. http://www-ai.ijs.si/MarkoBohanec/dexi.html

Buchheim, C., Jünger, M., Leipert, S.: Improving Walker's algorithm to run in linear time. *Lecture Notes In Computer Science* 2528, Revised Papers from the 10th International Symposium on Graph Drawing, Springer-Verlag, 344–353, 2002.

Dwyer, T., Koren, Y., Marriott, K.: Drawing directed graphs using quadratic programming. *IEEE Transactions on Visualization and Computer Graphics* 12(4), 2006.

Goldfarb, D., Idnani, A.: A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming* 27, 1–33, 1983.

Kennedy, A.J.: Drawing Trees. *Journal of Functional Programming* 6(3), 527–534, 1996.

Walker, J.Q. II: A node-positioning algorithm for general trees. *Software—Practice and Experience* 20(7), 685–705, 1990.

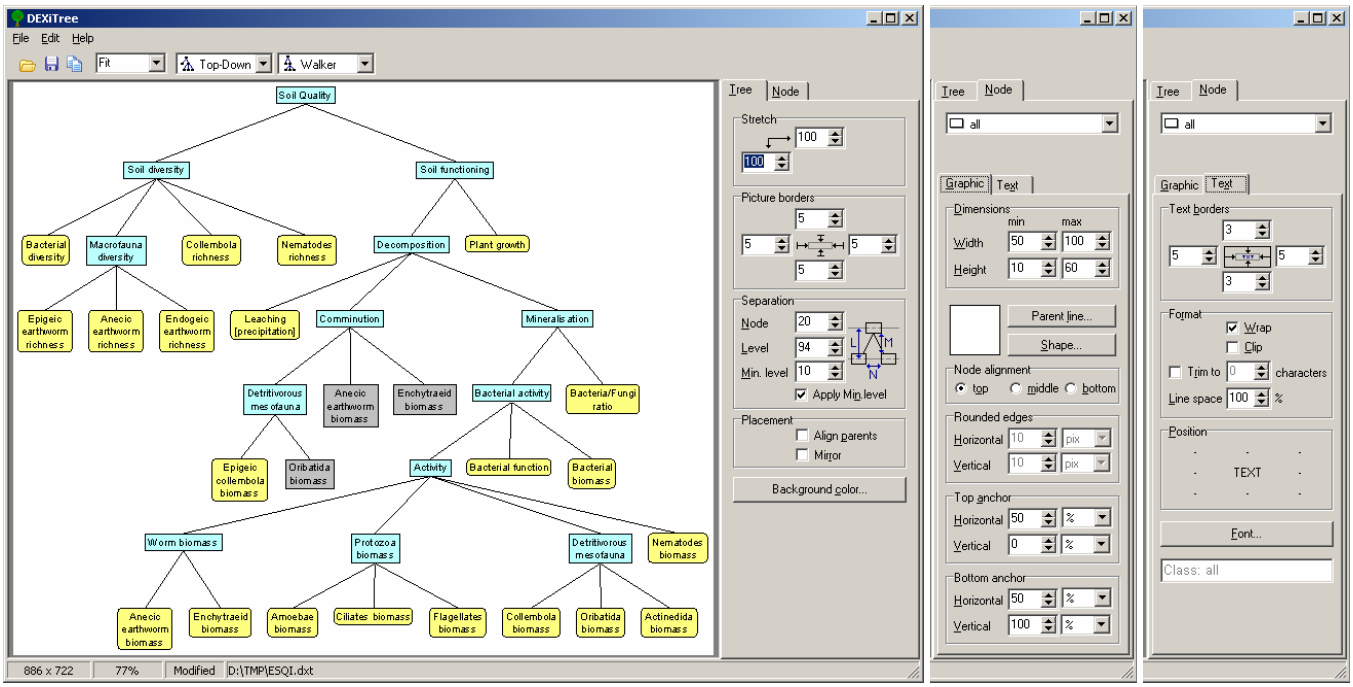

Figure 7: DEXiTree user interface: tree display (left) and tree-drawing parameters (right).