Minimum Size h-v Drawings

Peter Eades, Tao Lin [1] and Xuemin Lin [2]

Abstract

Trees are one of the most commonly used structures in computing, and many techniques for the visualization of trees are available. These techniques usually aim to find an aesthetically pleasing layout for a tree on a screen of limited size. This paper presents an algorithm for "h-v tree" drawing. The algorithm can be used to find a drawing of minimal "size", where "size" has a variety of definitions (including area). Two applications of the algorithm are explicitly presented.

1 Introduction

Many tree drawing algorithms [1, 3, 6, 8, 7, 10, 11, 12] have been developed to service needs in visualization and documentation systems. The aim of most of these algorithms is to draw a tree with in a limited space, subject to a variety of conventions. In this note we show how to obtain a minimal size drawing subject to some constraints defined below. We show how the algorithm can be applied to a visualization problem.

Suppose that T is a rooted binary tree with n nodes. A *drawing* π of T assigns a location $\pi_u = (x_u, y_u)$ to each node u of T. The drawing implicitly assigns the open line segment between π_u and π_v to each edge (u, v) of T. The drawing is *planar* if for each pair (u, v) , (s, t) of edges of T, the line segments representing (u, v) and (s, t) do not cross. The drawing is a *grid* drawing if x_u and y_u are integers for each node u.

A planar grid drawing of $T = (V, A)$ is a *h-v drawing* if (1) each edge is represented as either a horizontal straight line proceeding from the parent rightwards to the child, or a vertical straight line proceeding from the parent downwards to the child (that is, $x_u \leq x_v$, $y_u \geq y_v$ and either $x_u = x_v$ or $y_u = y_v$ for each edge (u, v) of T); and (2) for each vertex u with children v and w, the enclosing rectangles of the subtrees under v and w are disjoint.

A sample h-v drawing is in Figure 1.

[1]Department of Computer Science, University of Newcastle, Callaghan NSW Australia 2308. Fax: 61-49-216929, Phone: 61-49-216034, e-mail: eades@cs.newcastle.edu.au

[2]Department of Computer Science, University of Queensland, Queensland Australia 4072. e-mail: lxue@cs.uq.oz.au

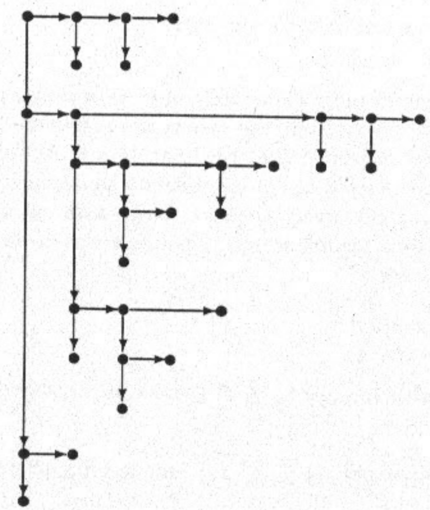

Figure 1: h-v tree

The *area* of a grid drawing is

$$(\max_{u \in V} x_u - \min_{u \in V} x_u)(\max_{u \in V} y_u - \min_{u \in V} y_u),$$

that is, the area of the smallest rectangle which encloses every node of the drawing.

It is shown in [2] that complete binary trees and Fibonacci trees have linear area h-v drawings, and every binary tree on n nodes has a h-v drawing with area $O(n \log n)$. These results are significant because a h-v drawing can be transformed easily into an "upward" drawing without asymptotically increasing the area (see [2]). Further, the results show that h-v drawings are feasible on limited screen/page areas.

In this note we present an algorithm which gives a minimum size h-v drawing for a rooted tree in time $O(n^2)$. The algorithm is based on results in [3]. Here the *size* of a drawing with width w and height h is $\psi(w, h)$, where ψ is a function which is nondecreasing in both coordinates. Examples of size functions are

1. *area:* $\psi(x, y) = xy$;
2. *perimeter:* $\psi(x, y) = 2(x + y)$;
3. *height for a given width:* $\psi(x, y) = y$ if $x < \eta$, $\psi(x, y) = \infty$ for $x \geq \eta$;

4. *minimum enclosing square*: $\psi(x, y) = \max(x, y)$,

The area size function is most commonly used as a measure for layout algorithms (especially for VLSI layout) but the others are sometimes more relevant for visualization purposes. For example, the third function is useful for the situation where the width of the page is fixed and it is necessary to minimize the height of the drawing; the fourth function effectively measures the maximum amount by which a drawing can be scaled up on a square screen. The algorithm in the next section can be applied to any size function.

A grid drawing is *reduced* if

- for each integer i with $\min_{u \in V} x_u \leq i \leq \max_{u \in V} x_u$, there is a node v with $x_v = i$; and
- for each integer j with $\min_{u \in V} y_u \leq j \leq \max_{u \in V} y_u$, there is a node v with $y_v = j$.

Since h-v drawings have only vertical and horizontal edges, one can assume without loss of generality that h-v drawings are reduced; we will make this assumption implicitly in the remainder of this paper. An immediate consequence of this assumption is that the width and height of a h-v drawing of a tree with n nodes are both at most $n - 1$.

2 The Algorithm

Suppose that π is a minimum size h-v drawing of a tree $T = (V, A)$. Denote the width and height of the smallest rectangle containing every node of the subtree under u by X_u and Y_u respectively.

If u is a leaf then clearly $X_u = Y_u = 0$.

If u has two children v and w, then there are essentially only four ways of arranging the subtrees T_v and T_w of v and w respectively, as in Figure 2.

Thus either

1. $(X_u, Y_u) = (X_v + X_w + 1, \max(Y_v + 1, Y_w))$, or
2. $(X_u, Y_u) = (X_v + X_w + 1, \max(Y_v, Y_w + 1))$, or
3. $(X_u, Y_u) = (\max(X_v + 1, X_w), Y_u + Y_w + 1)$, or

4. $(X_u, Y_u) = (\max(X_v, X_w + 1), Y_u + Y_w + 1)$.

depending on the arrangement.

If u has one child v, then there are only two possible arrangements: v is either one unit below u or one unit to the right of u. Thus either $(X_u, Y_u) = (X_v, Y_v + 1)$, or $(X_u, Y_u) = (X_v + 1, Y_v)$.

For each node u of T, define a set P_u as follows.

If u is a leaf, then $P_u = \{(0, 0)\}$.

If u has one child v, then P_u is the union of $\{(X_v, Y_v + 1) : (X_v, Y_v) \in P_v\}$ and $\{(X_v + 1, Y_v) : (X_v, Y_v) \in P_v\}$.

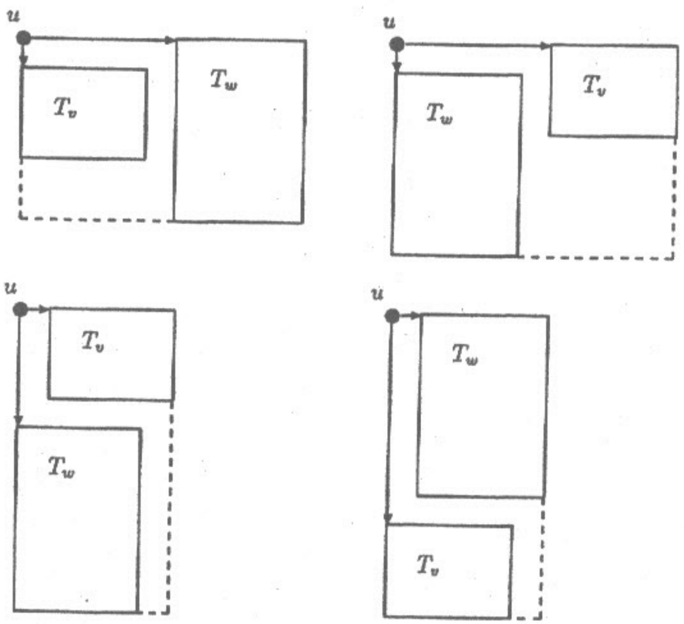

Figure 2: possible arrangements

If u has children v and w, then P_u is the union of the following four sets (each set corresponds to one of the four cases above):

1. $\{(X_v + X_w + 1, \max(Y_v + 1, Y_w)) : (X_v, Y_v) \in P_v, (X_w, Y_w) \in P_w\}$

2. $\{(\max(X_v+1, X_w), Y_u+Y_w+1) : (X_v, Y_v) \in P_v, (X_w, Y_w) \in P_w\}$
3. $\{(X_v+X_w+1, \max(Y_v, Y_w+1)) : (X_v, Y_v) \in P_v, (X_w, Y_w) \in P_w\}$
4. $\{(\max(X_v, X_w+1), Y_u+Y_w+1) : (X_v, Y_v) \in P_v, (X_w, Y_w) \in P_w\}$

It is clear that P_u represents the dimensions of all possible enclosing rectangles of a reduced drawing of the subtree under node u.

We say that a pair (c, d) of integers *dominates* (a, b) if $a \geq c$ and $b \geq d$. Basically, (c, d) dominates (a, b) if a rectangle with dimensions (a, b) will fit inside a rectangle with dimensions (c, d). The definition of "size function" ψ ensures that if (c, d) dominates (a, b) then $\psi(c, d) \geq \psi(a, b)$; thus if $(a, b) \in P_u$ dominates $(c, d) \in P_u$, then (a, b) will never be involved in an optimal layout and may be discarded.

An *atom* of a set S of pairs of integers is an element of S which dominates no other element of S; let A_u denote the set of atoms of P_u.

We will consider the case of computing A_u for a node u with two children v and w. Consider the first two possibilities for (X_u, Y_u) above. It is clear that one dominates the other and can be an atom of P_u; the dominated one has value

$$(X_v + X_w + 1, \min(\max(Y_v + 1, Y_w), \max(Y_v, Y_w + 1)))$$

Similarly, only one of the second two possibilities can be an atom of P_u. This gives the following Lemma.

Lemma 1 *Suppose that u has children v and w. Then A_u is the set of atoms of the union of*

1. $A_u^{HOR} = \{(X_v + X_w + 1, \min(\max(Y_v + 1, Y_w), \max(Y_v, Y_w + 1)) : (X_v, Y_v) \in A_v, (X_w, Y_w) \in A_w\}$
2. $A_u^{VER} = \{\min(\max(X_v + 1, X_w), \max(X_v, X_w + 1)), Y_u + Y_w + 1) : (X_v, Y_v) \in A_v, (X_w, Y_w) \in A_w\}$

□

We can compute A_u^{VER} with the following algorithm from [3], adapted from [9]. Here the sets A_v and A_w are represented as lists $A_v = \{(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)\}$ and $A_w = \{(c_1, d_1), (c_2, d_2), \ldots, (c_l, d_l)\}$, with $a_i > a_j$ and $c_i > c_j$ for $i < j$. Note that since these are sets of atoms and the first coordinates are in strictly decreasing order, $b_i < b_j$ and $d_i < d_j$ for $i < j$.


```
ALGORITHM VerticalMerge
i ← 1; j ← 1;
WHILE i ≤ k and j ≤ l DO
    p ← (min(max(a_i + 1, c_j), max(a_i, c_j + 1)), b_i + d_j + 1)
    IF p does not dominate the last element added to A_u^VER THEN add p to A_u^VER;
    IF a_i ≥ c_j THEN i ← i + 1 ELSE j ← j + 1;
```

This algorithm produces A_u^{VER} in decreasing order of first coordinate and increasing order of second coordinate. It clearly has linear time complexity. One can use a similar "Horizontal Merge Algorithm" to compute the list A_u^{HOR}. A third merge algorithm, to form A_u from A_u^{VER} and A_u^{HOR} and discard any dominated pairs, can be constructed easily.

These merge algorithms can be applied in a bottom-up traversal of a tree to compute A_r for the root r. A minimum size element of A_r can be chosen by a linear search.

Further, note that since every element of A_u is an atom, and no two atoms have the same width, we can deduce that $|A_u|$ is at most the number of nodes in the subtree under u. Our Theorem follows.

Theorem 1 *A minimum size h-v drawing of a binary tree with n nodes can be found in time $O(n^2)$.* □

More complex analysis of our algorithm gives better results for particular size functions; for example:

Theorem 2 *A minimum area h-v drawing of a binary tree with n nodes can be found in time $O(n\sqrt{n \log n})$.* □

These Theorems compliment the bounds established in [2]. However, the problem of finding a linear time algorithm for minimum size h-v drawings is open

3 Applications

The algorithm described above forms part of *Snake*, an object-oriented system developed by the second author for visualizing relational information. Here we mention two areas where Snake has used our algorithm in visualization.

Firstly, a system for animating LISP programs is described in [5]. This system uses diagrams of "S-expressions" such as:

$$((A(B))(((C)D)(E)F)(GH)). \quad (1)$$

The diagram composition rules for S-expressions in [5] essentially define the h-v drawing convention. Kamada gave variety of layouts using this convention for the expression (1). The layout of the expression (1) created by Snake using the algorithm described in the previous section is illustrated in Figure 3. This layout has the minimal size.

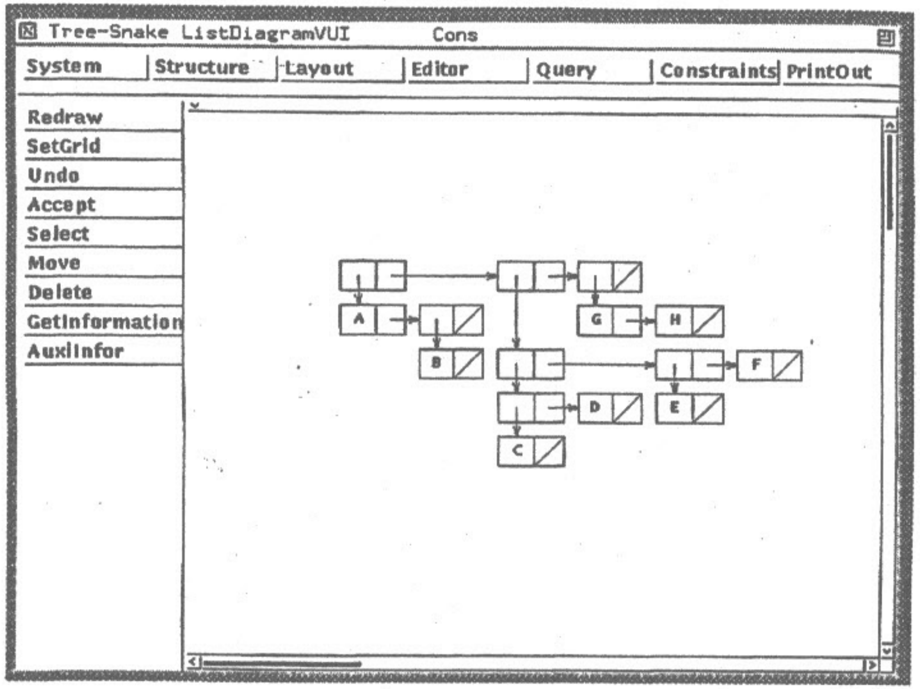

Figure 3: Layout for the LISP expression

We can also use this algorithm for creating the layout for some other kinds of diagrams. A layout for a proof diagram is in Figure 4; interested users can get more details of the concepts illustrated by this diagram from [4].

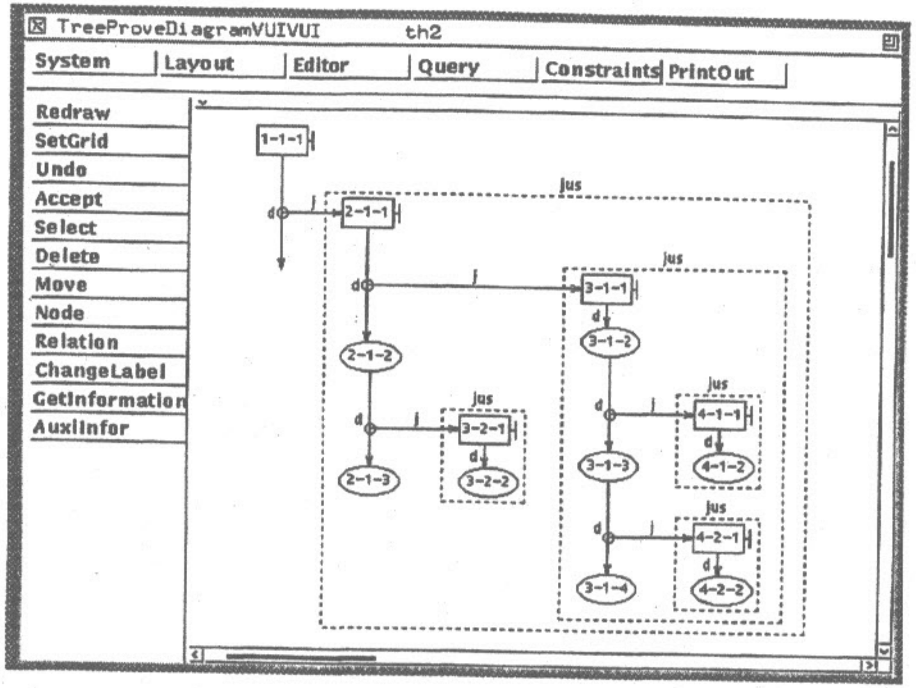

Figure 4: Proof Diagram

References

[1] A. Bruggemann-Klein and D. Wood. Drawings trees nicely with tex. Department of Computer Science, University of Waterloo, 1987.

[2] P. Crescenzi, G. DiBattista, and A. Piperno. A note on optimal area algorithms for upwrd drawings of binary trees. Technical Report RAP.11.91, Dipartimento di Informaticá e Sistemistica, Universita degli Studi di Roma "La Sapienza", 1991.

[3] P. Eades, X. Lin, and T. Lin. Two tree drawing conventions. (to appear in "Computational Geometry & Applications"), 1991.

[4] J. Han. *A Structural Model for Methodology-based Interactive Rigorous Software Development.* PhD thesis, University of Queensland, 1992.

[5] T. Kamada. *Visualizing Abstract Objects and Relations*, volume 5 of *Series in Computer Science.* World Scientific, 1989.

[6] J. Manning and M.J. Atallah. Fast detection and display of symmetry in trees. *Congressus Numerantium*, 1989.

[7] S. Moen. Drawing dynamic trees. Technical Report LiTH-IDA-R-87-24, Department of Computer and Information Science, Linkoping University, 1987.

[8] E. Reingold and J. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, **SE-7**(2):223–228, 1981.

[9] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, **57**:91–101, 1983.

[10] J. S. Tilford. Tree drawing algorithms. Master's thesis, Department of Computer Science, University of Illinois at Urbana Champaign, 1981.

[11] J. Vaucher. Pretty printing of trees. *Software Practice and Experience*, **10**(7):553 – 561, 1980.

[12] C. Wetherall and A. Shannon. Tidy drawings of trees. *IEEE Transactions on Software Engineering*, **SE-5**(5):514 – 520, 1979.