Compact Layout of Layered Trees

Kim Marriott **Peter Sbarski**

Clayton School of IT, Monash University, Australia.
{marriott,sbarski}@mail.csse.monash.edu.au

Abstract

The standard layered drawing convention for trees in which the vertical placement of a node is given by its level in the tree and each node is centered between its children can lead to drawings which are quite wide. We present two new drawing conventions which reduce the layout width to be less than some maximum width while still maintaining the essential layered drawing convention. These conventions relax the requirement that a parent must be exactly placed midway between its children, and instead make this a preference which can be violated if this is required for the layout to fit into the required width. Both drawing conventions give rise to a simple kind of quadratic programming problem. We give an iterative gradient projection algorithm for solving this kind of problem, and also a linear time heuristic algorithm. Our algorithms are practical: a tree with three thousand nodes can be laid out in less than a hundred milliseconds with either algorithm.

1 Introduction

Trees, i.e. connected acyclic graphs, occur in a wide variety of applications, including computer data structures, parse trees, hierarchical database models, phylogenetic trees, hierarchically organised file systems (for example, directories and files), decision trees, organization charts, family trees and biological classifications. Given the practical importance of trees it is not surprising that there has been considerable research into tree layout. The standard layered drawing convention of trees stipulates (Brüggermann-Klein & Wood 1989, Kennedy 1996) that:

1. the y-coordinate of each node corresponds to its level,
2. nodes on the same level are separated by a minimum gap,
3. edges do not cross each other,
4. each node is centered directly over its children,
5. the drawing is symmetrical with respect to reflection,
6. the trees are drawn compactly.

Wetherell and Shannon (Wetherell & Shannon 1979) proposed a linear time algorithm for layered drawing of binary trees, and this was improved by Reingold and Tilford (Reingold & Tilford 1981). Walker (Walker 1990) extended the Reingold-Tilford algorithm to handle n-ary trees, while Buchheim, Jünger and Leipert (Buchheim, Jünger & Leipert 2002) gave an improved version of Walker's algorithm with linear time. The Reingold-Tilford algorithm and the subsequent extensions are the standard methods for drawing rooted ordered trees.

However, a disadvantage of these algorithms is that the resulting drawings can be be quite uncompact as the real-world example in Figure 1 shows. In part for this reason, a number of other drawing conventions for trees have been suggested (di Battista, Eades, Tamassia & Tollis 1999). These include radial drawings in which the layers are mapped to concentric circles and hv-drawings in which the edges are either drawn horizontally to the right, or vertically to the bottom. The disadvantage of these other drawing conventions is that the structure and hierarchical nature of the tree is less clear, and the layout looks "unnatural".

We present two new drawing conventions which reduce the layout width to fit in some maximum drawing width while still maintaining the essential layered drawing convention. Both conventions relax the requirement that a parent must be exactly placed at the midway between its children, and instead make this a preference which can be relaxed if this is required for the layout to fit into some maximum width. The drawing conventions give rise to a simple kind of quadratic programming problem in which a quadratic objective must be minimized subject to minimum separation between nodes on each level, and the layout fitting in the maximum width. We give a gradient projection algorithm for solving this kind of problem, and also a linear time heuristic algorithm.

Our techniques have application to most real-world visualisation of trees since a significant disadvantage of the standard layered drawing convention is that even for moderate sized trees the drawing becomes quite large and cannot easily be viewed on a computer monitor or printed on a reasonable number of pages. Thus there is a need to find layouts which are more compact and which better untilise the space in a page or on a screen. Figures 2, 3 and 4 show examples of the layout produced by our algorithms for the example from Figure 1.

In the next section we review the standard algorithms for layout of layered trees. In Section 3 we present the new drawing conventions, Section 4 contains the algorithms for finding a layout satisfying the conventions, and Section 5 gives an empirical evaluation of their performance, while Section 6 concludes.

2 Background

The Reingold-Tilford algorithm (Reingold & Tilford 1981) uses a divide-and-conquer strategy to find a lay-

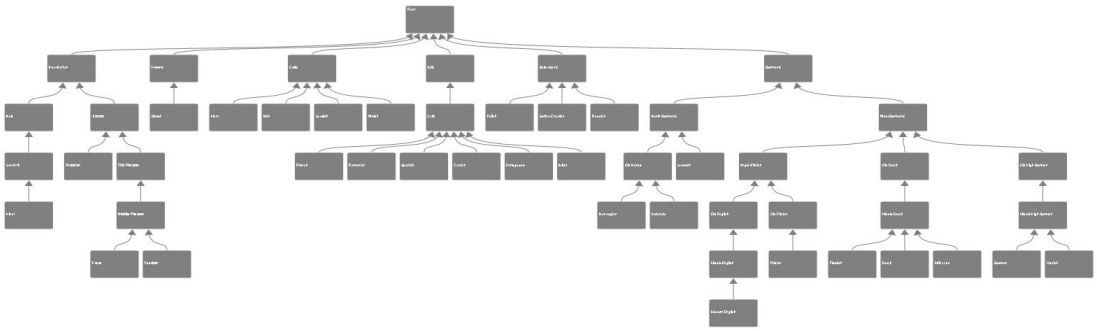


Figure 1: Example tree detailing the relationship between Indo-European languages drawn with the Walker Algorithm.

out. A single node is placed at 0, and a non-leaf node i is placed by a recursively laying out its sub-trees and then squashing the sub-trees as close as possible together and then placing i midway between the leftmost and rightmost child. Clever use of data structures allows this to be done in linear time.

Reingold-Tilford's algorithm can be modified to handle n-ary trees by traversing the children of a node left to right, placing and shifting the corresponding subtrees one after another. However, this approach may violate the aesthetic goal that the tree must be symmetric with respect to reflection since some branches can end up being clustered closely together as they are placed as close as possible to the left rather than being evenly spaced apart. Walker (Walker 1990) gave a modification of Reingold-Tilford's algorithm for n-ary tree layout which overcomes this problem. The contours of a node's sub-trees are traversed, and if there is an overlap, the right subtree is shifted by the minimum amount to the right. Any smaller subtrees situated between the overlapping left and right subtree are uniformly spaced between the two outer sub-trees. This guarantees that the symmetry is preserved. Unfortunately, the algorithm given by Walker has quadratic worst case complexity. Buchheim, Jünger and Leipert (Buchheim et al. 2002) improved Walker's algorithm to have linear time.

One nice property of Reingold-Tilford's and Walker's approach is that the drawing of a sub-tree does not depend on its context, and so isomorphic subtrees will have the same layout, and that drawings are symmetric with respect to reflection. However, because of this it will not always produce a layout of minimum width, as minimising the over all width may require that some sub-trees are drawn non-compactly. The problem is that minimizing the total width may mean that isomorphic subtrees need to be laid out differently in different parts of the tree as their global context is important.

If we do wish to minimize the total width while still placing a parent midway between its children then this can be modelled as a linear programming problem and solved in polynomial time (Supowit & Reingold 1983). However, in practice, the requirement that the parent is exactly midway between its children means that the minimal width layout is only very rarely narrower than that found using Reingold-Tilford's and Walker's approach. Therefore, in the next section we investigate drawing conventions in which the requirement to place each parent exactly midway between its children is relaxed. As we shall see this allows narrower layout.

3 Drawing Conventions that Relax Centering of Parents

Requiring that parents are midway between their children means that the drawing cannot be as narrow as possible. In this section we give two new drawing conventions for layered trees in which we must find the best layout for the tree that is no wider than some fixed width W. W might be set by the user interactively to try and better fit the layout within a display or print region. Of course, we require that W is greater than the cumulative width of the nodes on each level, since otherwise there is no drawing in which the nodes do not overlap.

We assume there are n nodes, $1, ..., n$ where node 1 is the root of the tree, and $1, ..., m$ levels with the root on level 1. We let w_i give the width of node i, and $rt(i)$ be the index of the node to the immediate right of i on the same level and let gap_i give the minimum gap between i and the node to its immediate right. For each level j, $lm(j)$ and $rm(j)$ respectively give the index of the leftmost and rightmost node on that level. We assume that nodes are numbered consecutively on each level.

We let x_i be the horizontal position of node i. We let $par(i)$ be the index of the parent of i, and $lc(i)$ and $rc(i)$ the index of the leftmost and rightmost child of i (they are the same if i has one child). We use the convention that if the node referred to by indexing function does not exist then the indexing function returns 0: For instance, $par(1)$ returns 0 as the root of the tree has no parent.

The first drawing convention is based on making trees compact by minimizing edge lengths. This has the affect of placing nodes at the average of their parent and children's positions subject to the non-overlap and minimum width constraints.

Drawing Convention: Minimizing distance between parents and their children (Min-Dist)
Minimize

$$\sum_{i=2}^{n}(x_{par(i)} - x_i)^2$$

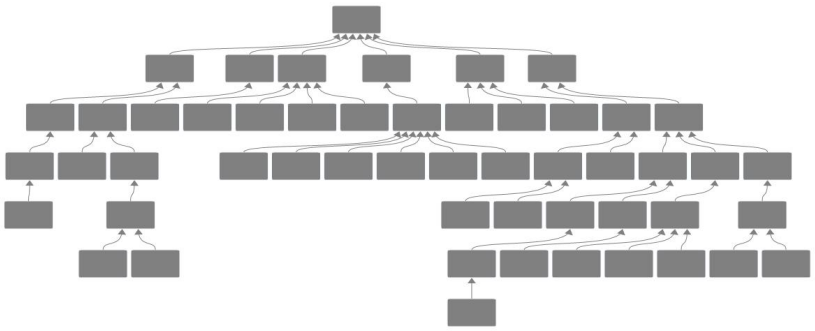
subject to: for all $i \in 2, ..., n$

- $x_i + \frac{w_i}{2} + gap_i \leq x_{rt(i)} - \frac{w_{rt(i)}}{2}$, if $rt(i) \neq 0$

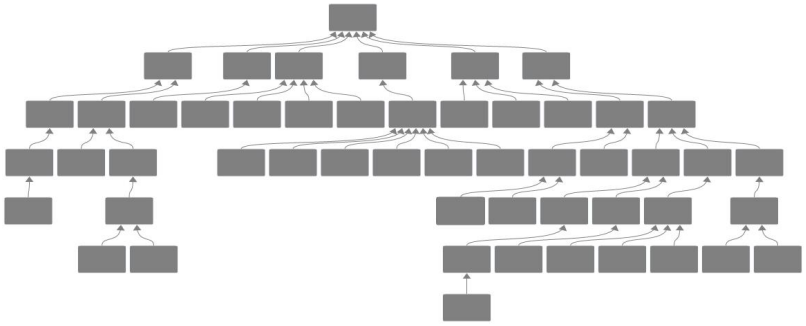
and for all $j \in 1, ..., m$,

- $0 \leq x_{lm(j)} - \frac{w_{lm(j)}}{2}$
- $x_{rm(j)} + \frac{w_{rm(j)}}{2} \leq W$.

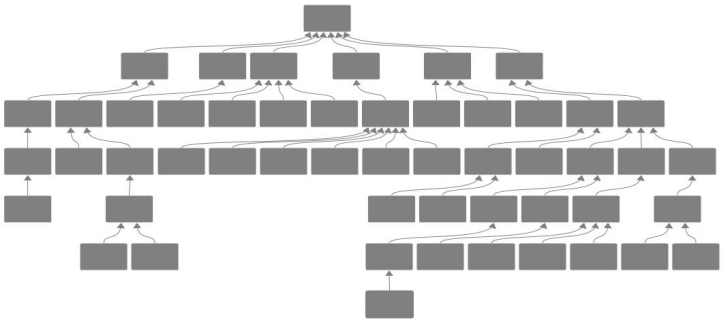
This is similar to the objective used in Gansner et al. (Gansner, Koutsofios, North & Vo 1993) to determine the placement of nodes on each layer in a Directed Acyclic Graphs (DAGs) drawn using the Layered drawing convention. Given a horizontal ordering



(a) Min-Dist: $W = W_{max}$



(b) Min-Dist: $W = W_{mid}$



(c) Min-Dist: $W = W_{min}$

Figure 2: Example tree from Figure 1 drawn with the Min-Dist drawing convention for different maximum widths W using the gradient projection algorithm.

of the nodes on each layer, their algorithm determines the x-coordinates by minimizing edge lengths between nodes. The main difference is that they model this as a linear problem and so minimize

$$\sum_{i=2}^{n} |x_{par(i)} - x_i|$$

rather than

$$\sum_{i=2}^{n} (x_{par(i)} - x_i)^2.$$

As pointed out in (Marriott, Moulder, Hope & Twardy 2005), using a linear objective may lead to unnecessary asymmetry as parents are not necessarily centred with respect to their children since the objective function uses absolute value. Thus, if the root of the tree has two children, it can be placed at any point between these without changing the penalty. Therefore, we prefer to use a quadratic objective function since in the above example this will prefer to place the root midway between its children.

However, in general the Min-Dist drawing convention will not place nodes midway between their children but rather at the average of their children and parent's horizontal position, since this minimizes the total horizontal extent of the edges. If placing a parent exactly between its leftmost and rightmost child is regarded as important we can extend the Min-Dist drawing convention by adding another term to the objective function to achieve this:

Drawing Convention: Placing parents midway between their children (Par-Midway)
Minimize

$$\sum_{i=2}^{n} (x_{par(i)} - x_i)^2 + \alpha \times \sum_{\substack{1 \le i \le n \ s.t. \\ rc(i) \neq 0}} \left(x_i - \frac{x_{lc(i)} + x_{rc(i)}}{2} \right)^2$$

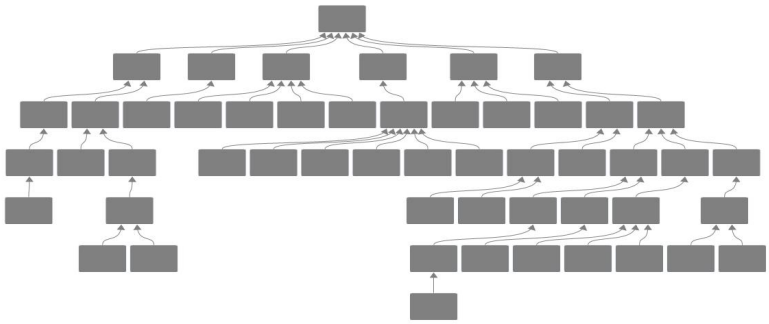
subject to: for all $i \in 2, ..., n$

- $x_i + \frac{w_i}{2} + gap_i \le x_{rt(i)} - \frac{w_{rt(i)}}{2}$, if $rt(i) \neq 0$

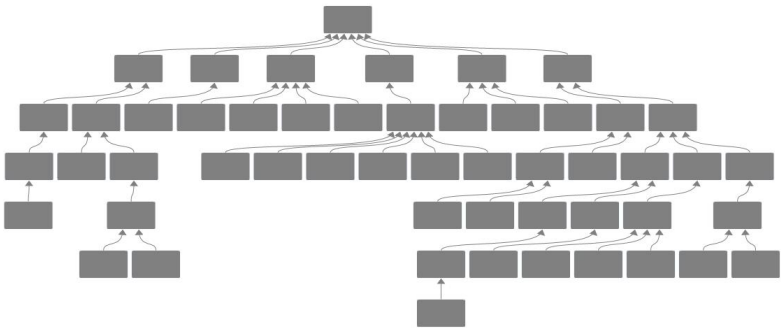
and for all $j \in 1, ..., m$,

- $0 \le x_{lm(j)} - \frac{w_{lm(j)}}{2}$
- $x_{rm(j)} + \frac{w_{rm(j)}}{2} \le W$.

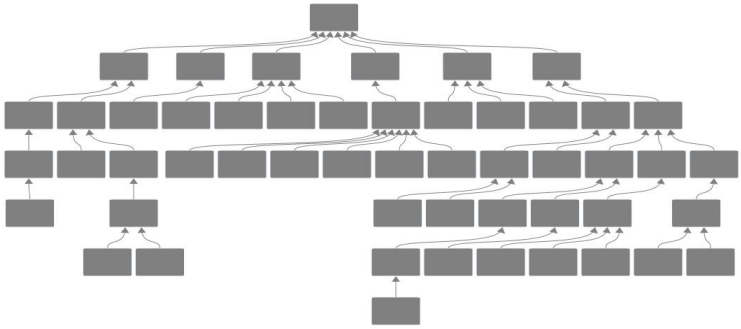
The scaling coefficient α specifies the relative importance of the two components of the objective. If α is equal to 0, then Par-Midway is identical to the Min-Dist drawing convention, but as α increases in



(a) Par-Midway: $W = W_{max}, \alpha = 1.0$



(b) Par-Midway: $W = W_{mid}, \alpha = 1.0$



(c) Par-Midway: $W = W_{min}, \alpha = 1.0$

Figure 3: Example tree from Figure 1 drawn with the Par-Midway drawing convention for different maximum widths W and a small value of α using the gradient projection algorithm.

value the resulting layout will tend to place parents midway between their children. It is also worth pointing out that if W is wider than the width of the layout obtained with Walker, then the layout obtained with Par-Midway for large α will be very similar to that obtained by the Walker algorithm. Of course, the main feature of interest is that W can be decreased down to the minimum width for the tree, and the two drawing conventions will "squish" the layout into that width.

Figures 2, 3 and 4 show examples of the layout produced by our algorithms for the example from Figure 1 and illustrate the effect of α and W on the layout. We give the layout for three widths: W_{max}, the width of the layout using the Walker algorithm; W_{mid}, the width of the layout of the narrowest possible layout; and W_{mid}, the average of W_{max} and W_{min}. We can see that the Min-Dist drawing convention always gives quite compact, narrow layout.

4 Layout Algorithms

In this section we investigate how to find layouts which satisfy the Par-Midway and Min-Dist drawing conventions. Both require minimising a quadratic objective function of form

$$\min_x x^T A x$$

subject to some separation constraints between nodes on the same level where a *separation constraint* c is of form $u + a \leq v$ where u, v are variables and a is the minimum gap between them or an upper or lower bound on a variable v of form $c \leq v$ or $v \leq c$.

For the Min-Dist drawing convention we have that A is A^{MD} where for each node i,

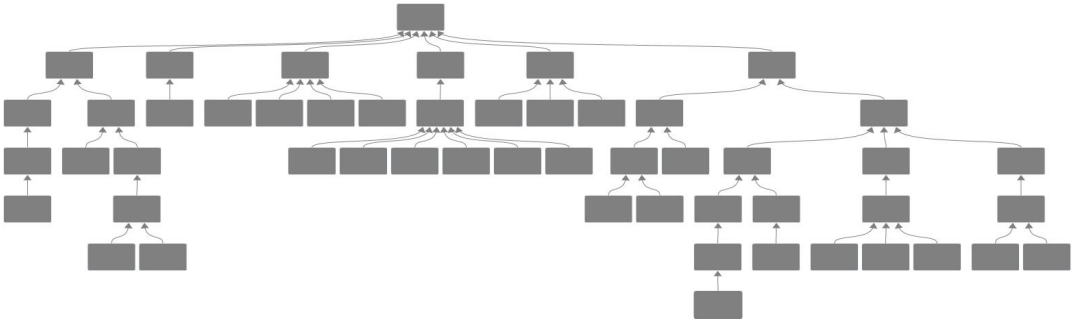
- $A^{MD}_{i,i} = deg(i)$ (where the degree, $deg(i)$, of i is the number of children of i plus the number of parents) and
- $A^{MD}_{i,par(i)} = A^{MD}_{par(i),i} = -1$

and all other entries in A^{MD} are zero.

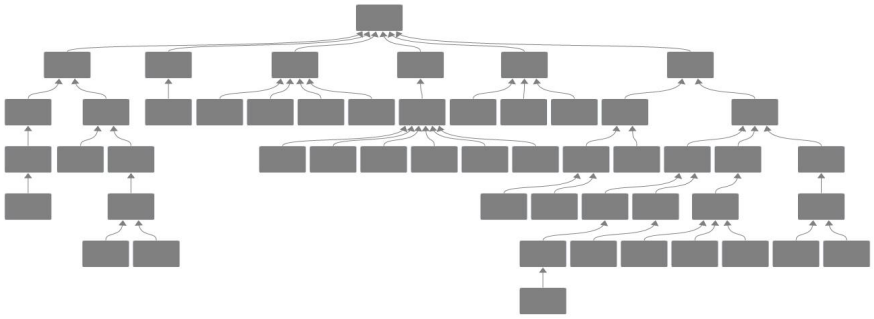
For the Par-Midway drawing convention we have that A is

$$A^{PM} = A^{MD} + \alpha \times \sum_{i=1}^{n} C^i$$

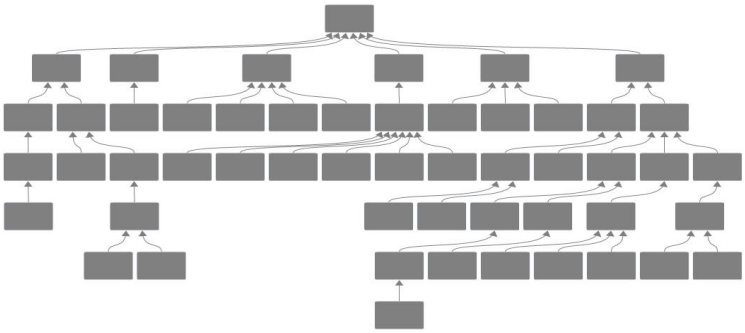
where for each node i,



(d) Par-Midway: $W = W_{max}, \alpha = 1.0E7$



(e) Par-Midway: $W = W_{mid}, \alpha = 1.0E7$



(f) Par-Midway: $W = W_{min}, \alpha = 1.0E7$

Figure 4: Example tree from Figure 1 drawn with the Par-Midway drawing convention for different maximum widths W for a very large α using the gradient projection algorithm.

- if i has no children, $C^i = 0$,
- if i has one child c, non-zero entries are

$$C^i_{i,i} = C^i_{c,c} = 1,$$
$$C^i_{i,c} = C^i_{i,c} = -1$$

- if i has leftmost and rightmost children l and r, the non-zero entries are

$$C^i_{i,i} = 1,$$
$$C^i_{l,l} = C^i_{r,r} = 1/4,$$
$$C^i_{i,l} = C^i_{l,i} = C^i_{i,r} = C^i_{r,i} = -1/2,$$
$$C^i_{l,r} = C^i_{r,l} = 1/4.$$

For both drawing conventions the matrix A is symmetric and positive semi-definite. Since separation constraints are linear this means that both conventions require solving a convex quadratic program.

4.1 Iterative Gradient Projection Methods

We now give an iterative *gradient-projection* algorithm (see Bertsekas (Bertsekas 1999)) for finding a layout for the above kind of problem. It is based on the gradient projection algorithm given in (Dwyer, Koren & Marriott 2006) but specialized to the particular case of trees. The algorithm, *gp_layout*, is shown in Fig. 5. This works by taking an initial layout, such as that obtained with the Walker algorithm, and then iteratively improving the placement of nodes. At each iteration, the direction of steepest descent, $-g$, is computed where g is the gradient $\nabla x^T Ax = 2Ax$ (actually $\frac{1}{2}g$ is computed, but multiplying the descent direction by a constant has no affect). Then the step size s along $-g$ from x that minimizes the objective function is determined and x is set to this new position. Since the new value of x may violate the separation constraints this is corrected by calling *project*, which returns the closest point $\bar{x}$ to x which satisfies the separation constraints, i.e. it projects x on to the feasible region. Finally, a vector d from the initial position $\hat{x}$ to $\bar{x}$ is computed and a decrease in the objective when moving in this direction is ensured by computing a second stepsize α which minimizes the objective in this interval.

While the algorithm given in Figure 5 describes a fairly standard gradient-projection approach, the procedure *project* is specific to our particular quadratic program. The main difficulty in implementing gradient-projection methods is the need to efficiently

procedure *gp_layout*(A, W)
  $x \leftarrow$ *initial_soln*()
  repeat
    $g \leftarrow Ax$
    $s \leftarrow \frac{g^T g}{g^T A g}$
    $\hat{x} \leftarrow x$
    $x \leftarrow \hat{x} - sg$
    $\bar{x} \leftarrow$*project*(x, W)
    $d \leftarrow \bar{x} - \hat{x}$
    $\alpha \leftarrow \min(-\frac{g^T d}{d^T A d}, 1)$
    $x \leftarrow \hat{x} + \alpha d$
  until $\|\hat{x} - x\|$ sufficiently small
  return x

procedure *project*(x, W)
  for $j \leftarrow 1$ to m **do**
    $g \leftarrow [gap_i \mid i \in lm(j), ..., rm(j) - 1]$
    $d \leftarrow [x_i \mid i \in lm(j), ..., rm(j)]$
    $x_{lm(j)}, ..., x_{rm(j)} \leftarrow$ *level_project*(d, g, W)
  end for
  return x

Figure 5: Gradient projection algorithm to layout a tree with n nodes on m levels s.t. the layout is no wider than W, nodes have a minimum separation and the position x for the nodes minimizes $x^T Ax$ where A is a symmetric positive-semidefinite matrix.

project on to the feasible region. That is, we must solve the quadratic problem

$$\min_y \sum_{i=1}^{n} (y_i - d_i)^2$$

subject to the same constraints on the variables y as the original problem.

Fortunately, the procedure *optimal_layout* given in (Marriott et al. 2005) can be used to project on to the feasible region. It solves the quadratic problem

$$\min_y \sum_{i=1}^{m} w_i \times (y_i - d_i)^2$$

subject to a separation constraint of form $y_i + g_i \leq y_{i+1}$ for each $i = 1, \ldots, m-1$ where g_i is the minimum separation between y_i and y_{i+1}, d_i is the desired position for y_i and w_i the importance of placing y_i at d_i.

The algorithm works by merging variables into larger and larger blocks of variables where a block is a sequence of variables with the minimum gap between each pair. The variables are processed left to right. When variable y_i is processed it is placed in a new block b at its desired location d_i. If the preceding block overlaps b it is merged with b and the desired position of the block is set to the weighted mean of the desired locations of variables in the block. This is repeated until b does not overlap the preceding block. The algorithm has linear complexity.

We can use *optimal_layout* to compute the projection for the nodes in a single layer: the only trick is that we add a dummy node at the beginning of the layer and one at the end whose desired values are 0 and W respectively, and whose weight is much higher than that of the other nodes. The two dummy nodes force the other nodes to fit in the desired width for the layout. The procedure *level_project* does this. By combining the projection for each layer, we have the overall projection. Since *optimal_layout* has linear time complexity, the overall projection has linear time complexity.

procedure *bottom_up_narrow*(W)
  $x \leftarrow$ *walker*()
  for $j \leftarrow m$ to 1 **do**
    $g \leftarrow [gap_i \mid i \in lm(j), ..., rm(j) - 1]$
    $d \leftarrow [des(i) \mid i \in lm(j), ..., rm(j)]$
    $x_{lm(j)}, ..., x_{rm(j)} \leftarrow$ *level_project*(d, g, W)
  end for
  return x

Figure 7: Linear time bottom-up algorithm to layout a tree with n nodes on m levels s.t. the layout is no wider than W and nodes have a minimum separation.

It follows from standard results on gradient projection and the fact that this is a convex optimisation problem that

Theorem 1 *gp_layout converges to a solution that minimizes $x^T Ax$ subject to the node separation and maximum width constraints.*

As we shall investigate in the next section, the algorithm is reasonably fast. The matrix A has only $O(n)$ non-zero entries. This means that as long as a sparse representation is used for A, each iteration takes only linear time since the projection step takes only linear time.

4.2 Bottom-up Algorithm

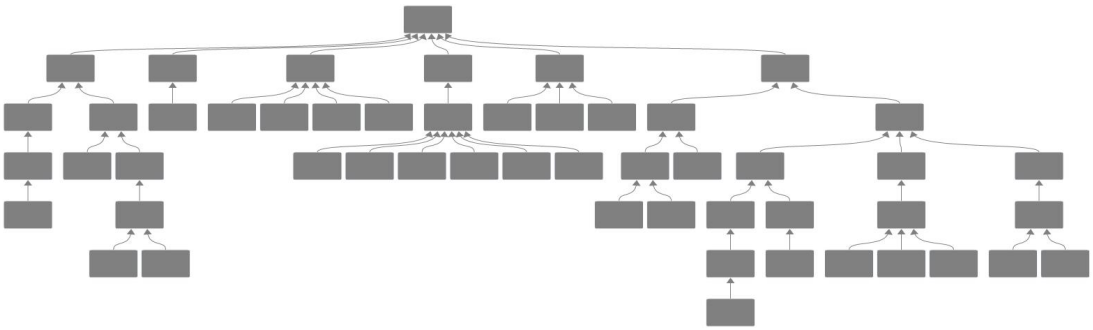
We have also investigated a one pass bottom-up approach to find a layout satisfying the width restriction. The algorithm is given in Fig. 7. The idea is quite simple. We first determine a layout using Walker's algorithm. Now we process the layers from the bottom up. For each layer j, we compute the desired position d_i of each node i in the layer using the function $des(i)$ where if i has children, $des(i)$ is their midpoint $(x_{lc(i)} + x_{rc(i)})/2$, otherwise $des(i)$ is the node's current position x_i. The new position, x_i, for each node i is computed by using *level_project* to project the desired values on to the closest solution satisfying the minimum width and node separation constraints for that level, effectively squishing them into the width W.

Figure 6 shows the layout for various values of W for the examples from Figure 1. It is instructive to compare them with the layouts shown in Figures 2, 3 and 4 obtained using the gradient projection algorithm to find a layout satisfying the Par-Midway drawing convention for $\alpha = \infty$, since the procedure *bottom_up_narrow* can be viewed as a heuristic for finding a layout satisfying this convention. As these examples show, it is not guaranteed to find the optimal solution, but, it will always find a solution satisfying the width and separation constraints, and it does this in linear time assuming the linear time version of Walker's algorithm is used.

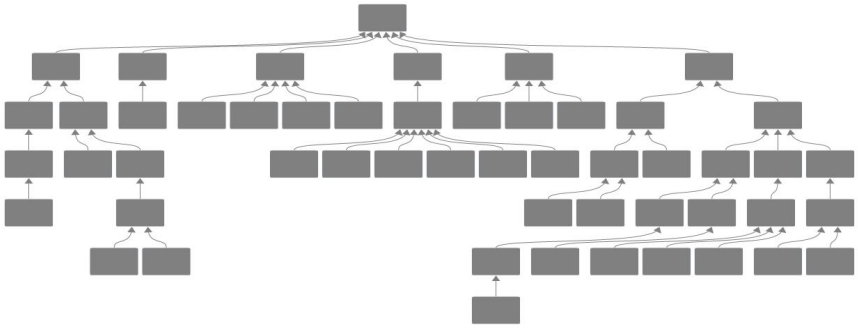
5 Evaluation

To evaluate the efficiency of our algorithms we laid out 10 random trees, with 100 to 3000 nodes. Times can be seen in Table 5.

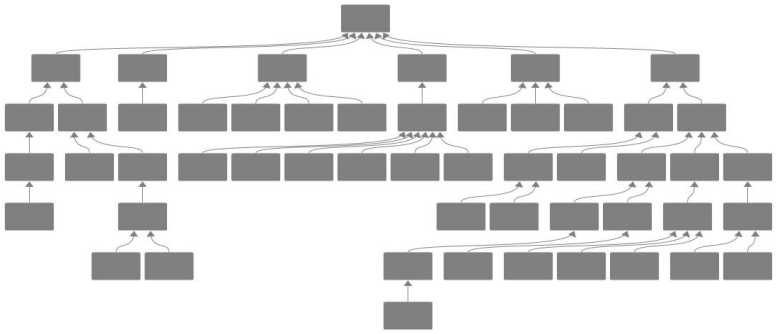
All experiments were run on a 1.83 GHz Intel Centrino with 1GB of RAM. The algorithms were implemented using Microsoft Visual C# Compiler version 8.00.50727.42 for Microsoft .NET Framework version 2.0.50727 under the Windows XP Service Pack 2 operating system. We used the original Walker algorithm



(a) $W = W_{max}$



(b) $W = W_{mid}$



(c) $W = W_{min}$

Figure 6: Example tree from Figure 1 drawn for different maximum widths W using the Bottom-up Layout Algorithm.

rather than the linear version: as the results show performance of the original algorithm is effectively linear and extremely fast.

We used our implementation of the Walker algorithm to compute the initial solution for both the bottom-up and the gradient projection algorithms. We did not include the time taken to compute this initial solution: this is simply the time for running Walker's algorithm. The tolerance for terminating the gradient projection algorithm was set to 0.002 of the width.

The speed of both the gradient projection algorithm and the bottom-up algorithm is very satisfying: even very large trees can be laid out in less than 200 milliseconds and the speed is comparable with that of Walker's algorithm, the standard layered tree layout algorithm. The main reason for the fast performance of the gradient projection algorithm is that the number of iterations is quite small since the algorithm quickly converges to the optimal layout.

6 Conclusion

The standard layered drawing convention for trees can lead to wide layouts because each parent must be placed exactly midway between its children. We have introduced two new drawing conventions for layered trees: *Min-Dist* in which the horizontal distance between parent's and children is minimised, and *Par-Midway* in which we minimize the horizontal distance between parent's and their children but also try and minimise the distance between a parent and the midpoint of its children. Thus both drawing conventions relax the requirement that a parent must be exactly placed at the center of its children, and instead effectively make this a preference which can be relaxed if this allows the tree to fit in a drawing of the desired width.

We have given an iterative gradient projection based algorithm for computing tree layouts using these two new drawing conventions. Experimental evaluation shows that the algorithm is fast enough for practical applications: a tree with three thousand nodes can be laid out in a few hundred milliseconds, which is only a few times slower than Walker's algorithm. We have also given a linear-time bottom-up algorithm for narrowing a layout produced by the Walker algorithm. This is somewhat faster, with similar speed to Walker's algorithm, and takes less than 100 milliseconds to layout a tree with over three thousand nodes. The layout quality is also quite good and similar to that of the gradient projection algorithm.

Nodes	*Levels*	*Walker*	*Min-Dist*		*Par-Midway* ($\alpha = 1.0$)		*Bottom-up*	
			W_{max}	W_{min}	W_{max}	W_{min}	W_{max}	W_{min}
32	9	18	16 (4)	17 (5)	15 (3)	15 (3)	4	4
49	9	18	18 (4)	20 (3)	14 (3)	16 (3)	4	4
52	7	20	18 (2)	25 (19)	16 (3)	17 (3)	4	4
84	16	21	19 (2)	21 (22)	17 (3)	21 (24)	5	4
220	16	21	43 (19)	69 (28)	39 (16)	70 (29)	6	5
365	28	22	17 (2)	25 (5)	17 (2)	29 (6)	7	7
673	22	24	29 (3)	35 (4)	20 (3)	27 (4)	11	11
1102	201	30	145 (38)	160 (40)	147 (38)	157 (39)	14	14
2101	31	33	66 (3)	79 (3)	63 (3)	65 (4)	26	21
3278	101	36	80 (3)	92 (3)	83 (3)	86 (3)	34	31

Table 1: Performance figures for 10 random trees. All timings are given in milliseconds. Figures in brackets specify the number of iterations. The gradient projection algorithm was used to compute the layout for the Min-Dist and Par-Midway drawing conventions.

References

Bertsekas, D. P. (1999), *Nonlinear Programming*, 2nd edn, Athena Scientific.

Brüggermann-Klein, A. & Wood, D. (1989), 'Drawing trees nicely with TeX', *Electronic Publishing* **2**(2), 101–115.

Buchheim, C., Jünger, M. & Leipert, S. (2002), Improving Walker's algorithm to run in linear time., *in* 'Graph Drawing', pp. 344–353.

di Battista, G., Eades, P., Tamassia, R. & Tollis, I. (1999), *Graph drawing: algorithms for the visualisation of graphs*, Prentice Hall.

Dwyer, T., Koren, Y. & Marriott, K. (2006), IPSep-CoLa: An incremental procedure for separation constraint layout of graphs, *in* 'Proc. IEEE Symp. on Information Visualisation (Infovis'06)'.

Gansner, E. R., Koutsofios, E., North, S. C. & Vo, K.-P. (1993), 'A technique for drawing directed graphs', *IEEE Transactions on Software Engineering* **19**(3), 214–230.

Kennedy, A. J. (1996), 'Drawing trees', *Functional Programming* **6**(3), 527–534.

Marriott, K., Moulder, P., Hope, L. & Twardy, C. (2005), Layout of Bayesian networks, *in* 'Australasian Computer Science Conference', Vol. 38.

Reingold, E. M. & Tilford, J. S. (1981), 'Tidier drawings of trees', *IEEE Transactions on Software Engineering* **7**(2), 223–228.

Supowit, K. & Reingold, E. (1983), 'The complexity of drawing trees nicely', *Acta Informatica* **18**.

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

Wetherell, C. & Shannon, A. (1979), 'Tidy drawings of trees', *IEEE Transactions on Software Engineering* **5**(5), 514–520.