Drawing Free Trees

Peter Eades

Department of Computer Science[1]
University of Newcastle
Newcastle, NSW
Australia, 2308
e-mail: eades@cs.newcastle.edu.au

Abstract. Trees are models of various structures in computing and it is not surprising that considerable efforts have been made toward effective drawing algorithms for them. This paper discusses drawing algorithms for "free trees", that is, trees with no special root. The aim of these algorithms is to provide a drawing which satisfies several aesthetic criteria, such as avoiding edge crossings, minimizing variance in edge length, and keeping vertices a reasonable distance apart. In this paper we measure the effectiveness of the algorithms presented by proving or disproving that they achieve such criteria.

1. Introduction

Trees are models of various structures in computing and it is not surprising that considerable efforts have been made to create effective drawing algorithms for them. Most interest has been in drawing *rooted* trees [6,13,18,20,22,26,27] because such trees are ubiquitous.

In this paper we consider drawing algorithms for "free trees", that is, trees with no special root. A drawing of a free tree is in Figure 1.

Such trees are frequently used in Combinatorial Mathematics, and occur in Computing as spanning trees and networks. But the most significant motivation for drawing them is that all general graph drawing systems include a module for drawing trees. The reason lies in the most common method for drawing a connected graph. A connected graph is partitioned into biconnected components and each biconnected component is drawn separately. Roughly speaking, the biconnected components form a tree structure (called the "block-cutpoint tree"), and we need a free tree drawing algorithm to "glue" the drawings of the biconnected components together. The method is described more fully in [17] and illustrated in Figure 2.

Compared to the wealth of literature on rooted trees, there little previous work on drawing free trees (the subject is mentioned in passing in [3,5,7,8,12,17], and more explicitly in [15,10]). The algorithms which are used in graph drawing systems are usually naive and attributed to folklore. In this paper we survey and discuss these algorithms and some of their variations.

[1] Written while the author was visiting the International Institute for Advanced Study of Social Information Science, Fujitsu Laboratories Ltd., 140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan

Bulletin of the ICA, Volume 5 (1992) 10-36

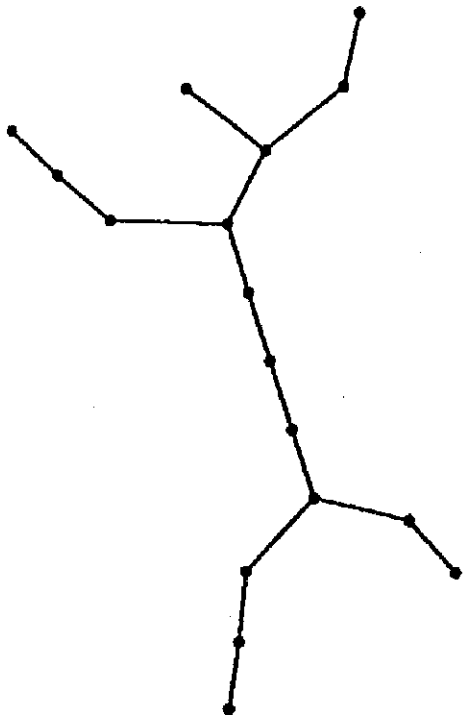


Figure 1: Example of a free tree

The aim of graph drawing algorithms is to provide a drawings which satisfy several *aesthetic criteria*, such as

- edge crossings are avoided.
- edges are as straight as possible.
- the length of edges is uniform.
- the distance between vertices is not too small.

There are many other criteria which are desirable; the above list is just a sample. In many cases it is possible to state such criteria in a precise mathematical way and prove or disprove that a particular algorithm achieves a particular criterion, or even to prove that achieving a particular criterion is NP-hard: see, for instance, [3]. Here we attempt to measure tree drawing algorithms against the kind of criteria above.

2. Terminology

Graph-theoretic terminology is mostly from [2]. The degree of vertex v is denoted by deg_v. The *graph-theoretic distance*, or *distance*, between two vertices u and v is the number of edges on the path joining u and v and is denoted by $d(u, v)$. The *diameter* of a tree $T = (V, E)$ is $\max_{u,v \in V} d(u, v)$.

The Euclidean distance between two points a and b in the plane is denoted by $||a - b||$.

All drawings discussed represent edges as straight lines between vertices, so a

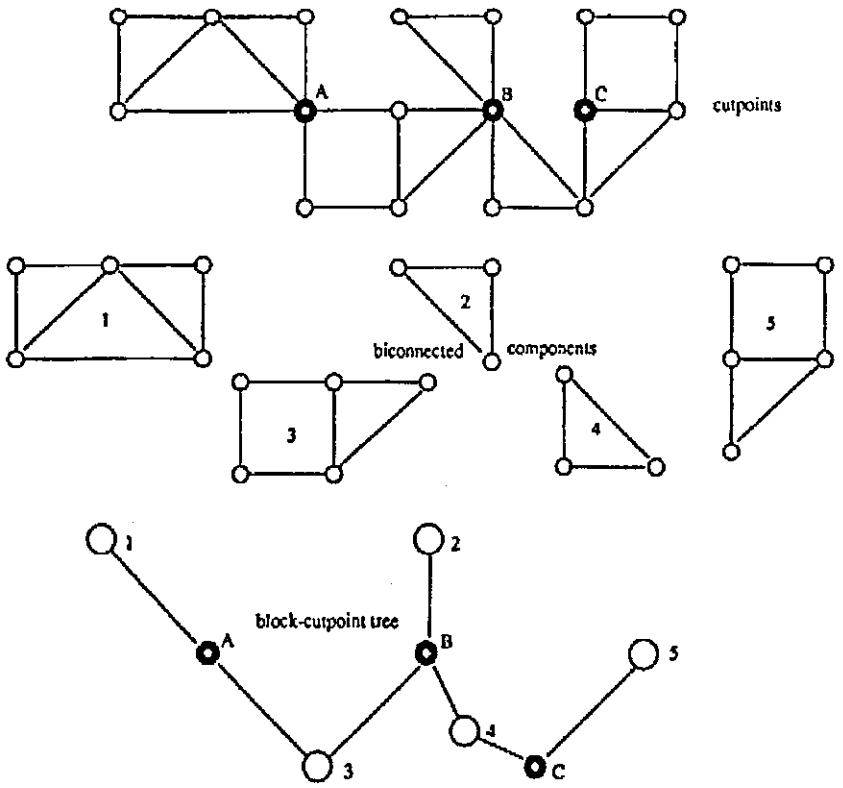


Figure 2: Block-cutpoint tree

drawing is completely determined by the positions of the vertices. Thus a *drawing* p of a tree $T = (V, E)$ is a function $p : V \rightarrow R^2$; $p(u)$ is the location of vertex u. Where confusion is unlikely we sometimes use the same symbol to denote both a vertex and the position of the vertex.

3. Radial Drawings

Radial drawings are well known in the folklore, probably originating in Ron Read's *GRAX* system (see [17]) of the late 1970s. Algorithms for such drawings are very similar to simple algorithms for drawing rooted trees. A root is chosen and drawn at the centre of the page. Nonroot vertices are arranged on concentric circles about the root. The subtrees of a vertex are drawn within *annulus wedges* of sizes determined by the sizes of the subtrees.

The basic concepts of this method are illustrated in Figure 3.

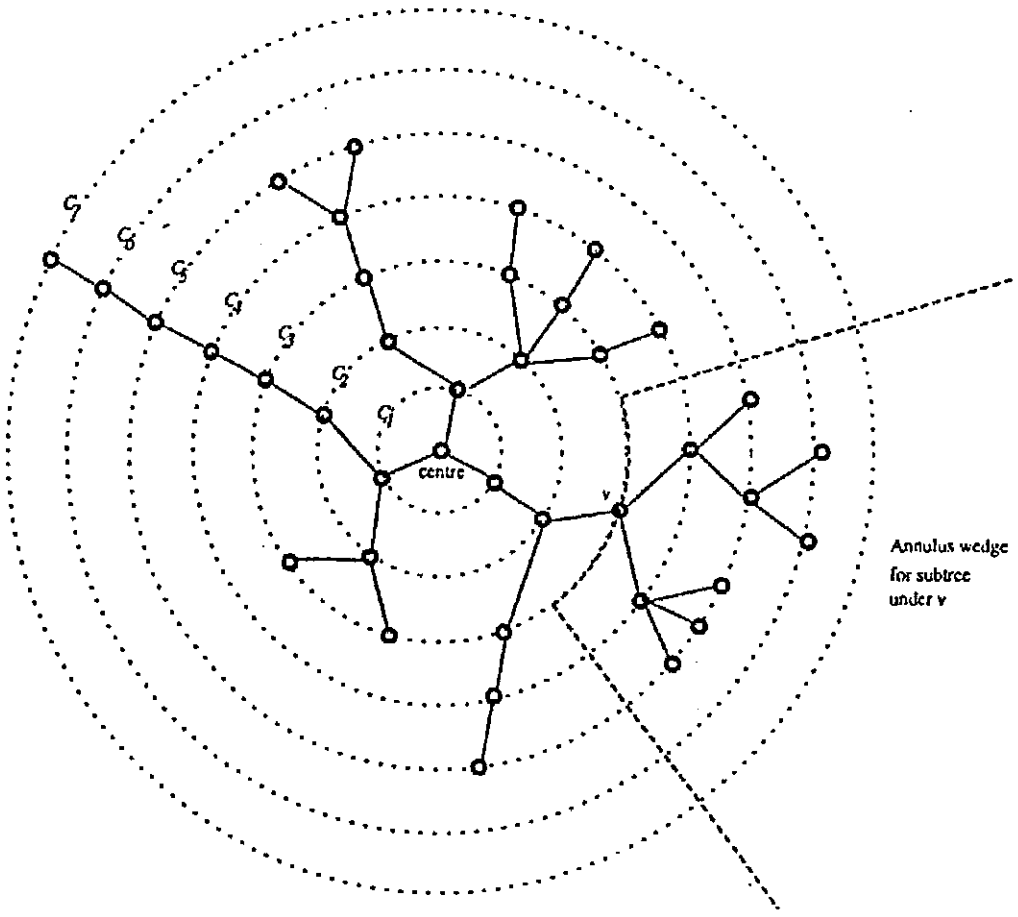


Figure 3: Radial algorithms

There are many variations of the basic method, depending on the choice of root, the radii of the circles, and the method for determining the size of the annulus wedge: see [1,7,15,17]. We shall describe a particular variation called *Algorithm* R_1 below.

The root is chosen to be the *centre* of the tree. A *central vertex* of a tree $T = (V, E)$ is a vertex c such that

$$\max_{v \in V} d(c, v)$$

is minimized. It can be shown [2] that a tree has either a unique central vertex or two adjacent central vertices. If T has a unique central vertex then this is the centre; otherwise the centre is the edge joining the two central vertices. The centre can be found in linear time using a simple leaf pruning algorithm: If T contains any nonleaves, then remove all the leaves of T to form a new tree T'; the centre of T is the same as the centre of T'.

The choice of a root means that we can use the terminology of rooted trees. For instance, the *depth* of a vertex v is the distance between v and its closest central vertex. The *subtree under* v for some vertex v consists of vertices and edges on paths between v and a leaf which do not contain a central vertex.

If the centre is a vertex then it is placed at the centre of the page; if the centre is an edge then it is drawn as a horizontal line of length δ (where δ is a constant) in the centre of the page.

Non-central vertices are assigned to concentric circles $C_1, C_2, \ldots, C_k$, centred at the centre of the page, where k is the largest depth of a leaf. If the centre is a vertex then the radius of C_i is $(i+1)\delta$, otherwise it is $(i+\frac{1}{2})\delta$. Vertices of depth i are placed on circle C_i. Thus, in a sense, vertices are placed as close as possible to the root.

The *width* $w(u)$ of a vertex u is the number of leaves in the subtree under u. It seems reasonable to draw the subtree under v in an annulus wedge with angle proportional to $w(v)$. However, this can lead to edge crossings, because an edge with endpoints within an annulus wedge of large angle can extend outside the annulus edge and intersect other edges, as in Figure 4.

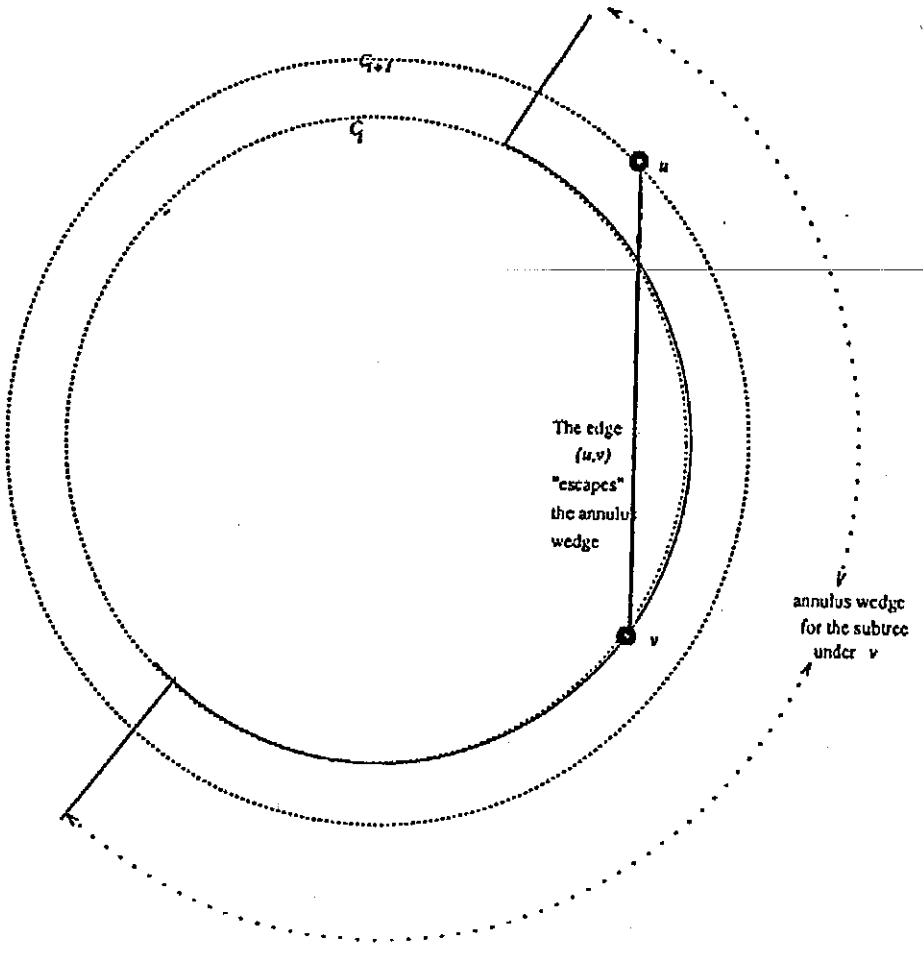


Figure 4: Edges escaping from an annulus wedge

The solution is to restrict the vertices to a convex subset of the wedge. Suppose that v lies on C_i, and that the tangent to C_i through v meets C_{i+1} at a and b as in Figure 5.

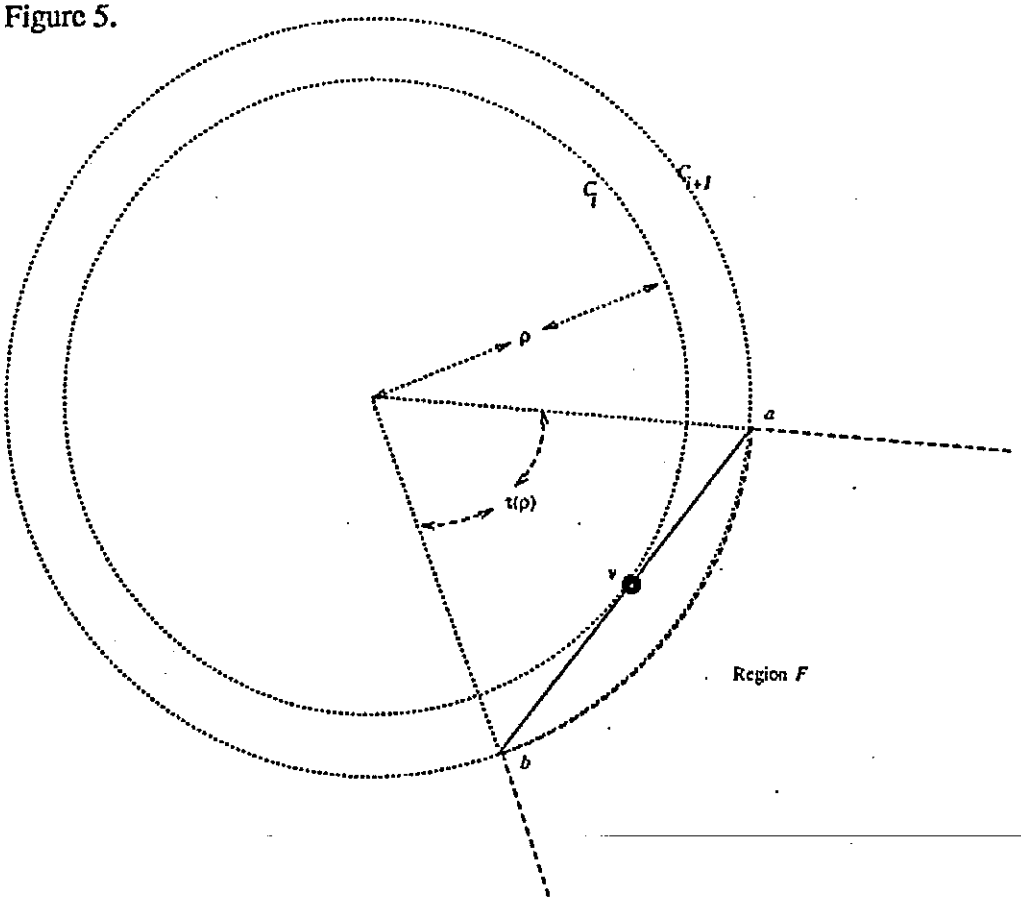


Figure 5: Convex subset of the wedge

Note that the unbounded region F in Figure 5, formed by the line segment ab, and the radii through a and b, is convex; thus any edge with endpoints in F cannot leave F. So we restrict the subtrees under v to lie within the region F. If the radius of C_i is ρ then we denote the angle between the radii through a and b by $\tau(\rho)$; note that

$$\tau(\rho) = 2\cos^{-1}\left(\frac{\rho}{\rho+\delta}\right).$$

The restriction to F effectively restricts the subtrees to lie within a wedge of angle $\tau(\rho)$.

The algorithm is stated below using this notation. The subroutine *DrawSubTree1*$(v, \rho, \alpha_1, \alpha_2)$ draws the subtree rooted at vertex v in the annulus wedge $\{(r,\theta) : r \geq \rho, \alpha_1 \leq \theta \leq \alpha_2\}$.

Algorithm R_1 ($T = (V, E)$: tree)
If the centre of T is a vertex v,
then DrawSubTree1$(v, 0, 0, 2\pi)$
else $(u, v) \leftarrow$ centre of T
DrawSubTree1$(u, \frac{\delta}{2}, \frac{3\pi}{2}, \frac{\pi}{2})$;
DrawSubTree1$(v, \frac{\delta}{2}, \frac{\pi}{2}, \frac{3\pi}{2})$

Algorithm DrawSubTree1$(v, \rho, \alpha_1, \alpha_2)$
1. $p(v) \leftarrow$ the point with polar coordinates $(\rho, \frac{\alpha_1+\alpha_2}{2})$.
2. if $\tau(\rho) < \alpha_2 - \alpha_1$
then $s \leftarrow \frac{\tau(\rho)}{w(v)}; \alpha \leftarrow \frac{\alpha_1+\alpha_2-\tau(\rho)}{2}$
else $s \leftarrow \frac{\alpha_2-\alpha_1}{w(v)}$ $\alpha \leftarrow \alpha_1$
3. for each child u of v do
DrawSubTree1$(u, \rho + \delta, \alpha, \alpha + s * w(u))$
$\alpha \leftarrow \alpha + s * w(u)$

Figure 6(a)–(d) shows samples of the output of R_1.

It is immediate that Algorithm R_1 runs in linear time.

Next we prove some that the output of Algorithm R_1 satisfies some of the desired aesthetic criteria.

The first result establishes planarity.

Theorem 1. *The output of Algorithm R_1 is planar.*

Proof: Note that every edge lies within an annulus between C_i and C_{i+1} for some i. However, it is impossible for two such edges to cross, because distinct subtrees are assigned to disjoint annulus wedges by Algorithm R_1. ∎

The next result bounds the variation in edge length.

Theorem 2. *Suppose that Algorithm R_1 returns a drawing $p : V \rightarrow R^2$ on a tree $T = (V, E)$ of diameter $D \geq 2$ and largest degree Δ, and*

$$L = \max_{(u,v)\in E} \|p(u) - p(v)\|,$$

$$l = \min_{(u,v)\in E} \|p(u) - p(v)\|.$$

Then

$$L \leq l\sqrt{1 + \frac{D}{2}(D-2)\left(1 - \cos\left(\left(1 - \frac{1}{\Delta - 1}\right)\cos^{-1}\left(1 - \frac{2}{D}\right)\right)\right)}$$

Proof: For notational convenience assume that $\delta = 1$. It follows that $l \geq 1$.

Now consider a vertex v on circle C_i of radius ρ. We shall compute a bound on the length of edges from v to its children.

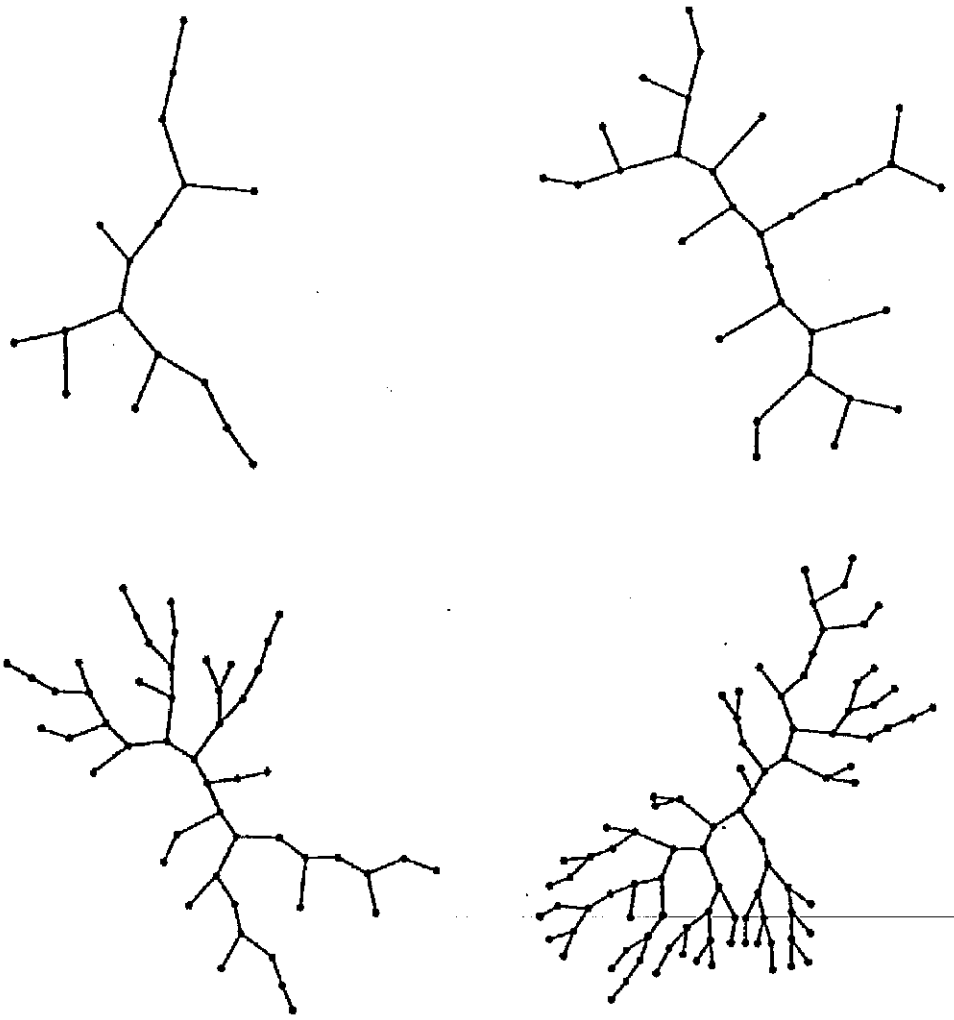


Figure 6:(a)–(d): Sample output from Algorithm R_1

The subtree under v is drawn within an annulus wedge with angle β; note from Algorithm R_1 that

$$\beta \leq \tau(\rho). \tag{1}$$

Suppose that Algorithm R_1 places a child u of v at a radial angle θ from v as in Figure 7.

Note that Algorithm R_1 ensures that

$$\theta \leq \left(1 - \frac{1}{w(v)}\right)\frac{\beta}{2}. \tag{2}$$

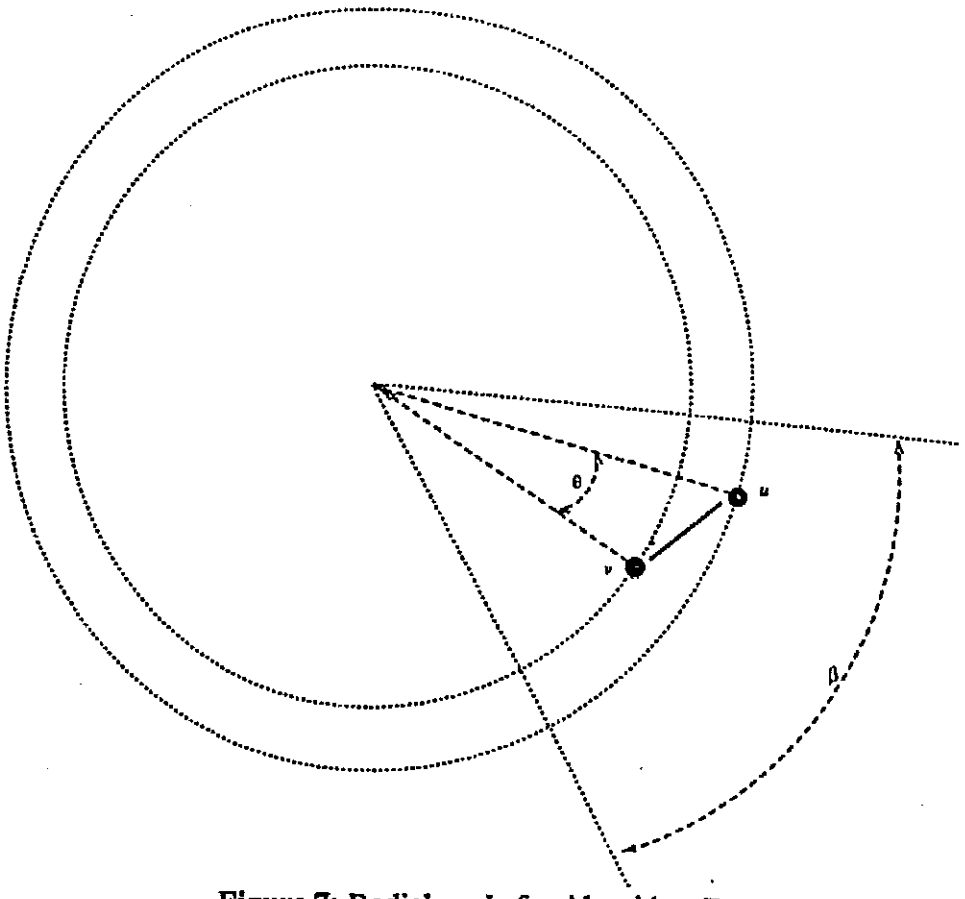


Figure 7: Radial angle for Algorithm R_1

A simple calculation reveals that

$$\|p(u) - p(v)\| = \sqrt{1 + 2\rho(\rho+1)(1-\cos\theta)}. \tag{3}$$

Using (1) and (2) we obtain

$$\|p(u) - p(v)\| = f(\rho, w(v)) \tag{4}$$

where

$$f(\rho, w) = \sqrt{1 + 2\rho(\rho+1)\left(1 - \cos\left(\left(1 - \frac{1}{w}\right)\tau(\rho)\right)\right.}. \tag{5}$$

Note that f is increasing in ρ. Now suppose that v is a vertex of maximum degree Δ. Since there are w leaves in the subtree under v, and vertices have at most $\Delta - 1$ children, there must be a descendant of v on the circle $C_{i+\lceil \log_{\Delta-1} w(v)\rceil}$, which has radius $\rho + \lceil \log_{\Delta-1} w(v)\rceil$. Thus the diameter D of T satisfies

$$D \geq 2(\rho + \lceil \log_{\Delta-1}(w)\rceil). \tag{6}$$

A tedious argument shows that the function f in equation (6) satisfies

$$f(\rho, w) \leq f(\rho + \lceil \log_{\Delta-1}(w) \rceil - 1, \Delta - 1). \tag{7}$$

The result follows using inequalities (6) and (7). ∎

The bound in Theorem 2 is worst for graphs with high diameter and a vertex of high degree near the perimeter: an example is in Figure 8.

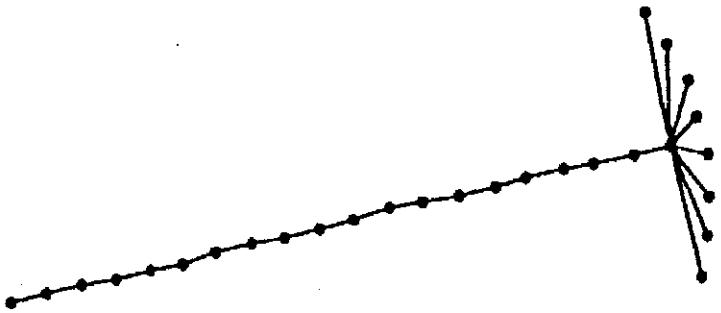


Figure 8: Worst case for edge lengths for Algorithm R_1

The following tabulation of D, Δ, and the maximum value of L/l in Theorem 2 gives an idea of the size of the bound.

D	Δ	max L/l
10	3	1.7
50	3	3.6
50	5	5.3
100	10	8.9
100	18	9.4

A simple consequence of Theorem 2 is helpful in giving a rough understanding of the length of edges.

Corollary 1. *Suppose that Algorithm R_1 gives a drawing $p : V \to R^2$ for a tree $T = (V, E)$ of diameter $D \geq 2$, and*

$$L = \max_{(u,v)\in E} \|p(u) - p(v)\|,$$

$$l = \min_{(u,v)\in E} \|p(u) - p(v)\|.$$

Then

$$L \leq l\sqrt{D-1}$$

∎

We can also show that the vertices of in an R_1 drawing are separated by a distance which depends on the diameter. We measure the ratio of this minimum separation distance to the maximum distance between two vertices.

Theorem 3. *Suppose that a tree of diameter $D \geq 2$ and w leaves is drawn by Algorithm R_1. Define*

$$K = \max_{u,v \in V} ||p(u) - p(v)||,$$

$$k = \min_{u,v \in V, u \neq v} ||p(u) - p(v)||.$$

Then

$$k \geq \min \left(1, 3 \sin \left(\frac{1}{w-1} \cos^{-1}\left(\frac{1}{3}\right)\right)\right) \frac{K}{D}$$

Proof: Again assume that $\delta = 1$. The Theorem clearly holds for $w = 2$, so we only consider the case $w \geq 3$.

Since the entire drawing is enclosed in a circle of diameter D, we have

$$K \leq D \tag{8}$$

If u and v are on separate circles, then

$$||p(u) - p(v)|| \geq 1$$

and the Theorem holds.

Now suppose that v is on a circle of radius ρ_v and the angle of the wedge assigned to v is β_v. Denote $\rho_v \sin(\frac{\beta_v}{2})$ by a_v; note that for any two vertices u and v on the same circle,

$$||p(u) - p(v)|| \geq a_u + a_v. \tag{9}$$

We will show that for every vertex v,

$$a_v \geq \frac{3}{2} \sin \left(\frac{1}{w-1} \cos^{-1}\left(\frac{1}{3}\right).\right) \tag{10}$$

The Theorem then follows from (8), (9) and (10).

Firstly note that either

$$\beta_v = \frac{2\pi w(v)}{w} \tag{11}$$

or for some ancestor z of v,

$$\beta_v = \frac{\tau(\rho_z) w(v)}{w(z)}. \tag{12}$$

In case (11), we have

$$a_v = \rho_v \sin\left(\frac{2\pi w(v)}{w}\right) \geq \sin\left(\frac{\pi}{w}\right)$$

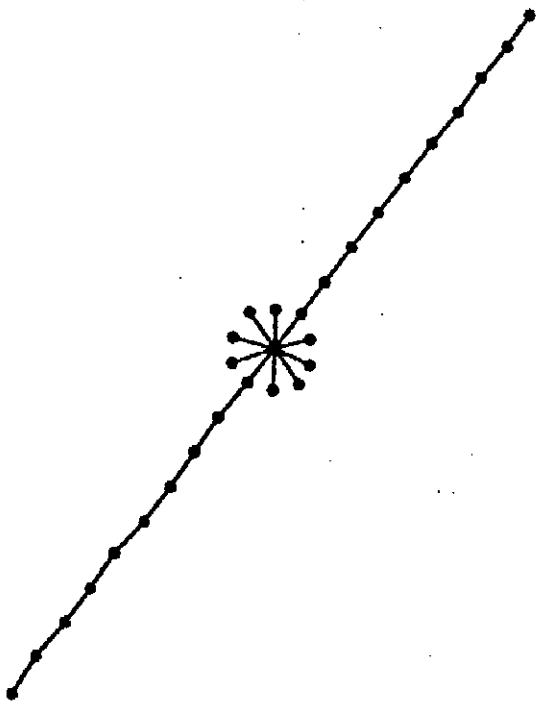


Figure 9: Worst case for vertex separation for Algorithm R_1

But a simple check reveals that for $w \geq 3$,

$$\sin\left(\frac{\pi}{w}\right) \geq \frac{3}{2}\sin\left(\frac{1}{w-1}\cos^{-1}\left(\frac{1}{3}\right)\right). \tag{9}$$

For case (12) observe that since $w(u) \geq 1$ and it is impossible for every leaf to be in the subtree under z,

$$\beta_v \geq \frac{\tau(\rho_z)}{w-1}. \tag{10}$$

Also, since z is an ancestor of v, $\rho_z \leq \rho_v - 1$ and thus $\tau(\rho_z) \geq \tau(\rho_v - 1)$. Thus

$$\beta_v \geq \frac{\tau(\rho_v - 1)}{w-1}, \tag{11}$$

and it follows that

$$a_v \geq \rho_v \sin\left(\frac{\tau(\rho_v - 1)}{2(w-1)}\right) = \rho_v \sin\left(\frac{1}{w-1}\cos^{-1}\left(\frac{\rho_v - 1}{\rho_v}\right)\right). \tag{12, 13}$$

But the right hand side of inequality (13) is increasing with ρ_v, and the minimum value of ρ_v is $\frac{3}{2}$; the inequality (10) follows. ∎

The bound in Theorem 3 is worst for graphs with high diameter and many leaves: an example is in Figure 9.

The following tabulation of D, w, and the minimum value of k/K in Theorem 3 gives an idea of the size of the bound. The Corollary below also gives a simple expression of the bound.

D	w	K/k
20	50	0.0065
20	100	0.0032
50	100	0.0020
50	200	0.0010

Corollary 2. *Suppose that Algorithm R_1 gives a drawing $p : V \to R^2$ for a tree $T = (V, E)$ of diameter $D \geq 2$ with w leaves, and*

$$K = \max_{u,v \in V} \|p(u) - p(v)\|,$$
$$k = \min_{u,v \in V, u \neq v} \|p(u) - p(v)\|.$$

Then

$$k \geq \min\left(1, \frac{3.6}{w-1}\right)\frac{K}{D}.$$

∎

All the edges in the drawing are straight lines. To make it easy to follow paths in the drawing, it is desirable that paths follow a straight line as well. We can show that although paths of vertices of degree two are not always straight in R_1 drawings, long paths are mostly straight.

Theorem 4. *Suppose that v is a vertex of degree 2, in the output of Algorithm R_1 on T, and the degree of at least one neighbor of v is 2. Then the edges incident with v form a straight line.* ∎

A minor variation of Algorithm R_1 is sometimes useful. Basically, we place each vertex v on the circle C_{k-i} where k is the maximum depth of a vertex in the tree and i is the distance from v to the closest leaf. To write this algorithm concisely, we denote the maximum distance from a vertex v to a leaf in the subtree under v by $h(v)$. Note that h can be computed in linear time. The result is algorithm R_2 below.

Algorithm R_2
If the centre is a vertex v,
then DrawSubTree2$(v, 0, 0, 2\pi)$
else $(u, v) \leftarrow$ centre of T
    DrawSubTree2$(u, (h(u) + 0.5)\delta, \frac{3\pi}{2}, \frac{\pi}{2})$
    DrawSubTree2$(v, (h(u) + 0.5)\delta, \frac{\pi}{2}, \frac{3\pi}{2})$

Algorithm DrawSubTree2$(v, \rho, \alpha_1, \alpha_2)$
1.place v at the point with polar coordinates $(\rho, \frac{\alpha_1+\alpha_2}{2})$.
2.if $\tau(\rho) < \alpha_2 - \alpha_1$
then $s \leftarrow \frac{\tau(\rho)}{w(v)}$; $\alpha \leftarrow \frac{\alpha_1+\alpha_2-\tau(\rho)}{2}$
else $s \leftarrow \frac{\alpha_2-\alpha_1}{w(v)}$ $\alpha \leftarrow \alpha_1$
3.for each child u of v do
    DrawSubTree2$(u, h(u), \alpha, \alpha + s * w(u))$
    $\alpha \leftarrow \alpha + s * w(u)$

Figure 10(a)–(d) shows samples of the output of R_2.

Planarity of the output can be established in a similar argument as for Algorithm R_1. The length of edges varies more widely for Algorithm R_2 than for R_1, but R_2 tends to have better minimum distance between vertices.

Bernard [B] has noted that a radial layout similar to R_1 can be used to draw trees symmetrically. It is comparatively easy to find automorphisms of trees (see, for example, [9]), and all automorphisms preserve the centre and the depth of vertices. Further symmetry oriented radial algorithms are presented in [10,15].

4. Spring Drawings

Spring algorithms use a concept of *energy* η defined for each drawing of a graph. The energy is defined so that smaller values of η indicate "nicer" drawings. The algorithms try to find drawings with η as small as possible. A particular spring algorithm is defined by a choice of energy function η and a method for minimizing η.

The simplest spring algorithm is from [5]. Intuitively adjacent vertices are connected with logarithmic strength "springs", and non-adjacent vertices repel each other with a force governed by an inverse square law. Note that a small energy indicates that the distance between adjacent vertices is about one unit and non-adjacent vertices are reasonably far apart. A drawing with a locally minimal energy may be found by a steepest descent method: an initial layout is chosen, then the layout is progressively improved by moving each vertex to a position with less energy. Trees drawn with this algorithm are shown in Figure 11(a)–(d).

An often noted problem with this approach is that it is possible for a planar graph to give a nonplanar drawing, even for trees: see [4,5,8]. For example, Figure 12 shows a drawing with locally minimal energy and many crossings.

One can overcome this problem by choosing the initial layout to be planar. We

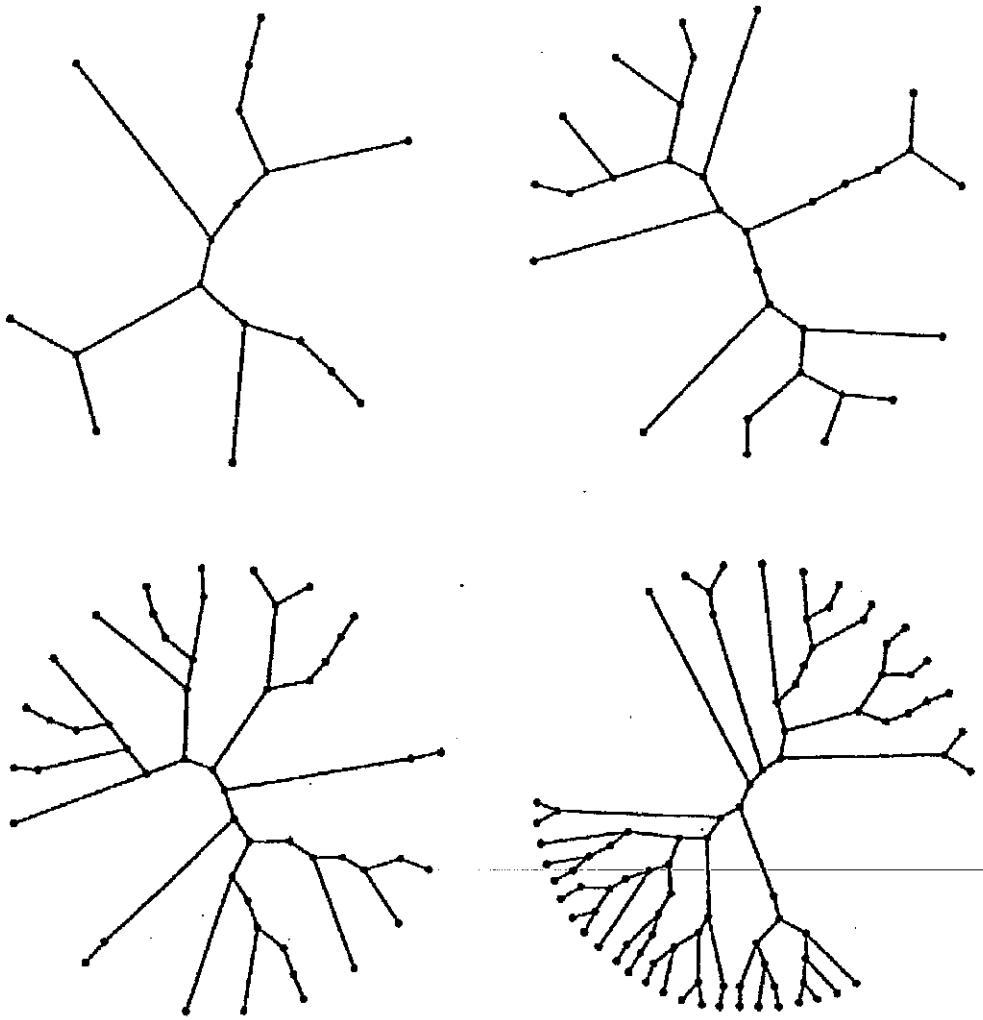


Figure 10(a)–(d): Sample output from Algorithm R_2

conjecture, but have been unable to prove, that this approach gives a planar drawing.

Conjecture. *Suppose that a drawing of a tree is obtained by a simple spring algorithm described above, using the the output of Algorithm R_1 as the initial layout. Then the drawing is planar.*

In fact it seems to be very difficult to prove properties of spring algorithms: with the exception of [14], most research in this area is informal, using extensive examples rather than mathematical proof to support claims. However, we we can show that another spring algorithm adapted from [23,24] gives planar drawings. Suppose that the leaves of a tree drawing are fixed in position, and edges are re-

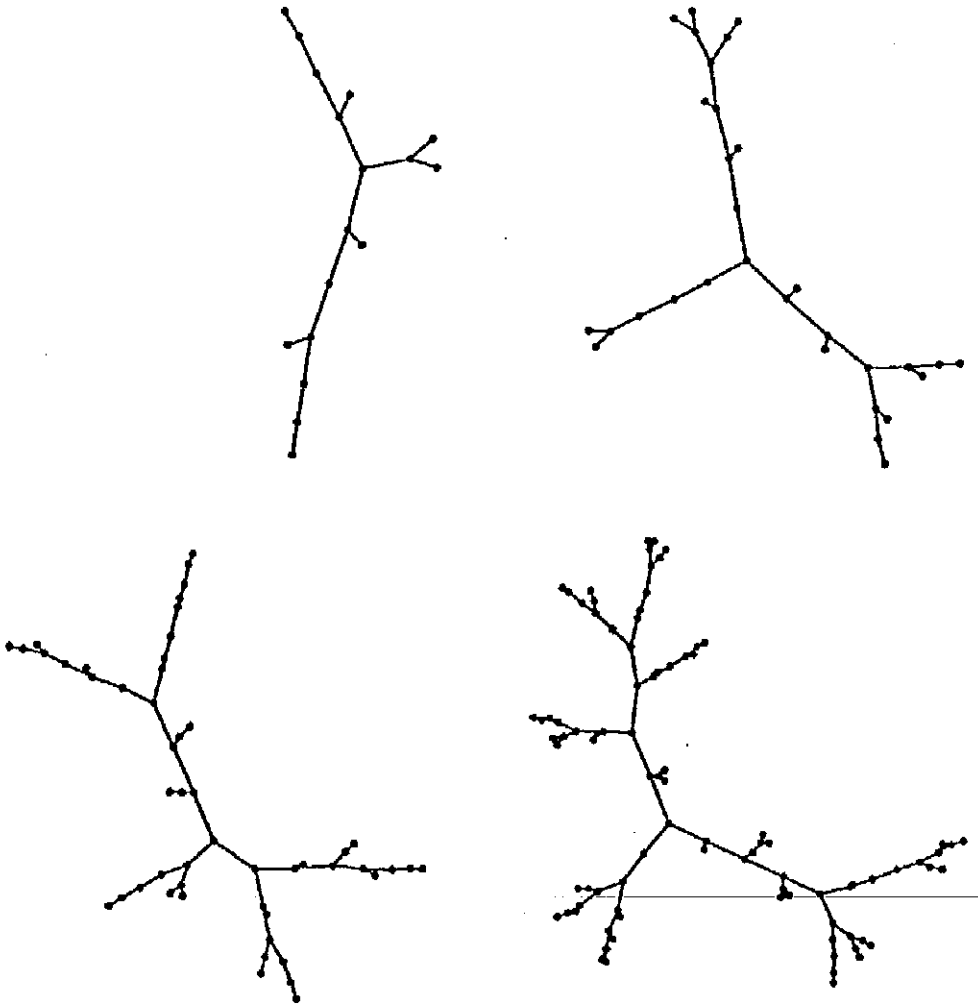


Figure 11:(a)–(d): Sample output from simple spring algorithm

placed by lengths of a kind of elastic cord. This elastic cord exerts a force on its endpoints proportional to the square of distance between the endpoints, so that the force acting on a vertex v is

$$\sum_{(u,v)\in E} \|p(u) - p(v)\|^2.$$

The sum of these forces is minimized when the partial derivatives are zero, so a minimum energy layout requires that

$$p(v) = \frac{1}{deg_v} \sum_{(u,v)\in E} p(u) \qquad (14)$$

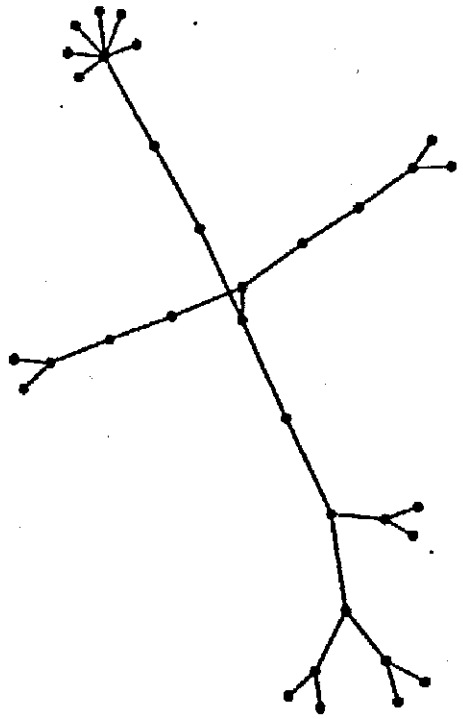


Figure 12: Poor result from simple spring algorithm

for each vertex v. The equations (14) simply say that each nonleaf vertex is located at the barycentre or average of the locations of its graph-theoretic neighbors.

The positions of the leaves can be fixed by one of the two algorithms R_1 and R_2 in the previous section. The equations may be solved by a simple numerical iteration (see [17]). For instance, we can define *Algorithm* T_1 as follows:

Algorithm T_1
Algorithm DrawSubTree2(v, p, α_1, α_2)
1.Apply Algorithm R_1
2.Repeat until error is small:
for each nonleaf v do $p(v) \leftarrow \frac{1}{\deg v} \sum_{u \in N_v} p(u)$

We can similarly define *Algorithm* T_2 using R_2 instead of R_1 to fix the leaves. Results of Algorithms T_1 and T_2 are in Figure 13.

Theorem 5. *The output of Algorithm* T_2 *is planar.* ∎

Proof: Consider the graph formed from a tree T by joining leaves in a cycle in depth-first order. This is a planar 3-connected graph, and if it is drawn by Algorithm T_2, the leaves form a convex polygon. It follows from Tutte's Theorem [23] that the drawing is planar. ∎

We believe, but have been unable to prove, that this claim also holds for T_1.

Conjecture 2. *The output of Algorithm* T_1 *is planar.*

Both algorithms give straight line paths of vertices of degree 2.

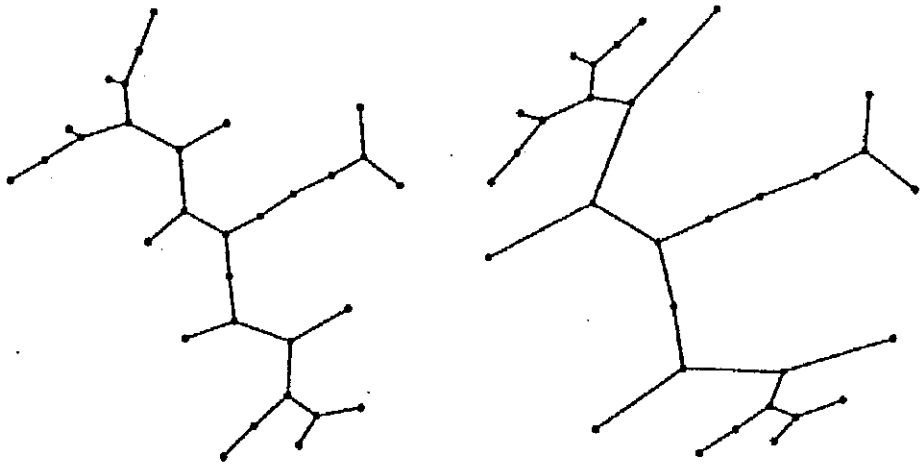


Figure 13: Sample output from T_1 and T_2

Theorem 6. *Suppose that v is a vertex of degree 2. Then, for both Algorithms T_1 and T_2, the edges incident with v form a straight line.* ∎

Proof: This is a simple consequence of the equations (14). ∎

The minimum distances are sometimes better for Algorithms T_1 and T_2 than for R_1 and R_2, but often they are worse, as one can see from Figure 14.

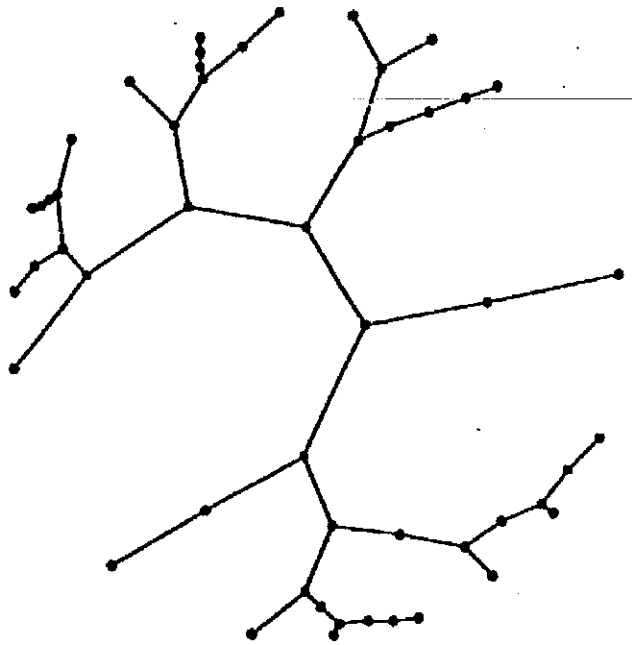


Figure 14: Poor result from T_2

The problem is that many vertices gather close together at the leaves. This can be overcome to some extent by weighting the average in the last line of Algorithm

T_1 above, that is, replace the last line by

$$p(v) \rightarrow \sum_{u \in N_v} t_{uv} p(u)$$

where the weights t_{uv} chosen so that or each v,

$$\sum_{u \in N_v} t_{uv} = 1.$$

For example, so that vertices may move closer to the centre of the diagram (to avoid situations such as in Figure 14) we can define:

$$t_{uv} = \frac{h(u)}{\sum_{w \in N_v} h(w)}$$

where $h(u)$ denotes the maximum distance of u a leaf in the subtree under u (where the root is a central vertex as in the previous situation). The result of applying this to the graphs in Figure 14 is in Figure 15.

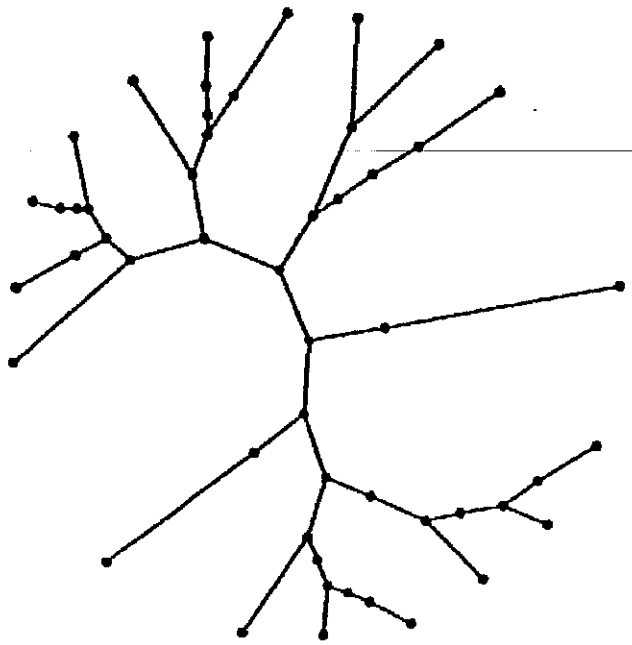


Figure 15: Sample from weighted version of T_2

Note that the problem of drawing a tree with fixed leaf positions so that edge lengths are all the same is NP-hard [28].

Kamada [10] describes another spring algorithm aimed at radial drawings of trees, based on [11]. Intuitively, every pair u, v of vertices is joined by spring s_{uv} with Hooke's Law strength and the zero energy length of s_{uv} is $d(u,v)$. A radial drawing is used as an initial drawing, and the energy is lowered subject to the vertices remaining on their original concentric circles about the centre. Intuitively, the vertices are allowed to "slide" around their circles, but not to leave the circles. In this way, it seems that the planarity of the radial drawing is preserved (but this property has not been proved).

Other variations [4,8,11,23,24] cover a range of energy functions. In particular, the energy function defined in [4] encodes variety of aesthetics, including the minimization of edge crossings; however, to minimize energy, slow simulated annealing process is necessary.

An attractive aspect of these algorithms is that their output tends to appear symmetric. It can be shown [14], under reasonable assumptions on the energy function, that there is a drawing with locally minimal energy which displays a maximal set of automorphisms of the input graph as symmetries.

5. H-tree Drawing

The discussion in this section is limited to *binary* trees; that is, trees where no vertex has degree larger than 3.

The H-tree layout for a complete binary tree, illustrated in Figure 16, is well known in the VLSI layout folklore (though originally due to Shiloach [19]) as a method for obtaining a compact layout of a complete binary tree.

Esposito [7] noted that the H-tree has some attractive properties as a "nice" drawing. It is planar, and all edges are either horizontal or vertical straight lines. However, for binary trees which are not complete, the layout can be very poor, as in Figure 17.

Here the vertices near the leaves are exponentially close to each other. In this section we describe a simple variation which keeps a reasonable minimum separation between vertices.

For this variation, an edge is chosen as a root. We chose an edge whose removal separates the tree into components such that the difference in the number of vertices in each component is minimal.

For each vertex v, the subtree under v is drawn in a rectangle with v inside. Four parameters N_v, E_v, S_v, W_v are computed for this rectangle representing the distances from the root v to the perimeter of the rectangle in directions north, east, south and west respectively, as illustrated in Figure 18.

If v has children u and w and the parent of v is to the west of v, then the following equations may be used to compute the parameters of v from the parameters of u

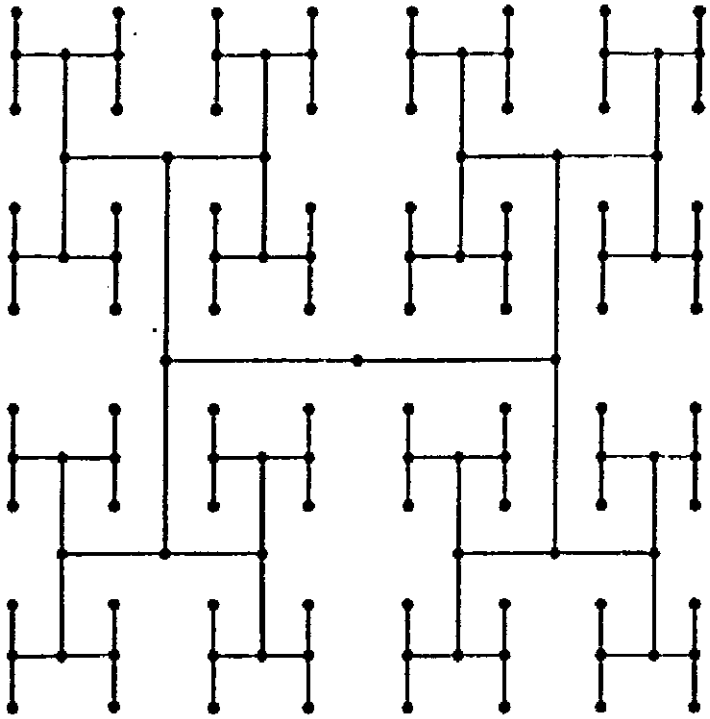


Figure 16: Complete *H*-tree

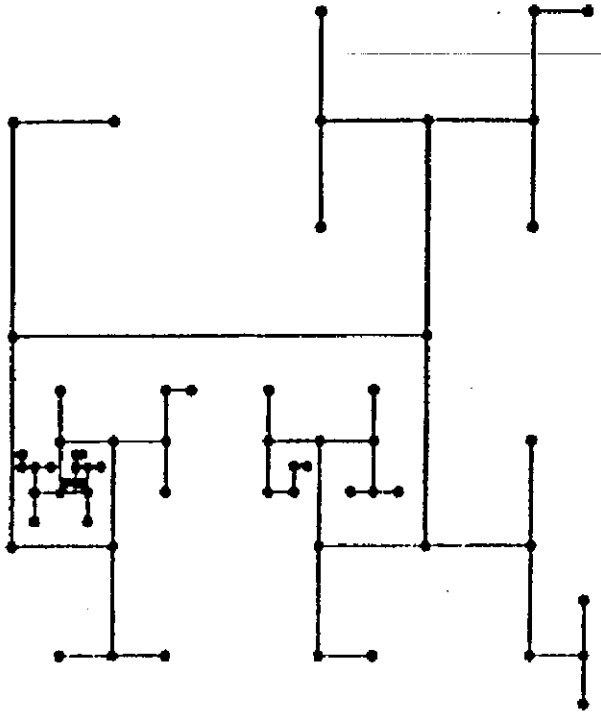


Figure 17: Poor result from the usual H-tree algorithm

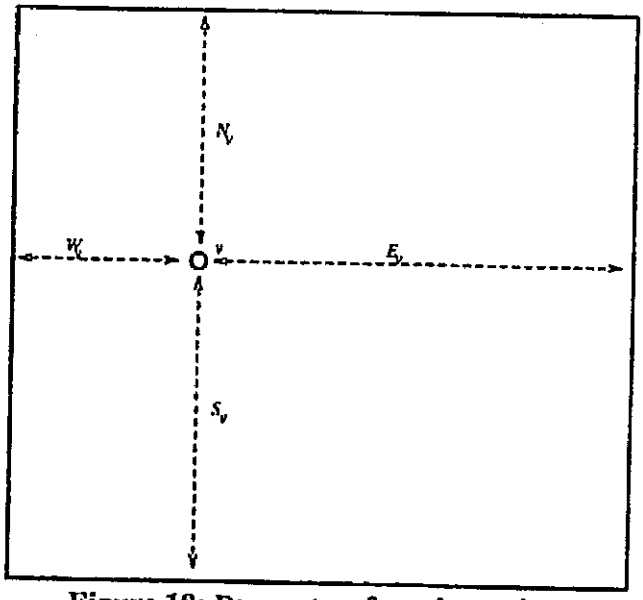


Figure 18: Parameters for subtree size

and w:

$$N_v = N_u + S_u + \delta$$
$$Ev = \max(E_u, E_w)$$
$$S_v = N_w + S_w + \delta$$
$$W_v = \max(W_u, W_w),$$

where δ is a constant. If v has a single child u and the parent of v is to the west of v, then:

$$N_v = N_u$$
$$E_v = E_u + W_u + \delta$$
$$S_v = S_u$$
$$W_v = 0.$$

These equations are illustrated in Figure 19.

Similar equations may be used if the parent of v is to the north, east, or south of v.

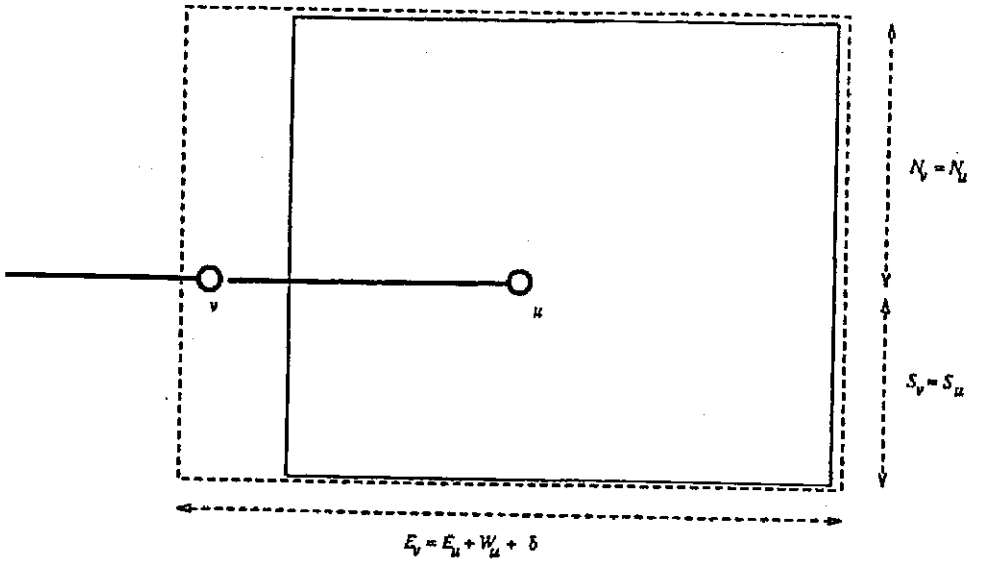
If v is a leaf then all of N_v, E_v, S_v, W_v are zero. A bottom up traversal may be used to compute N_v, E_v, S_v, W_v for nonleaves v. Once these parameters are computed the location of the vertex can be determined by a top down traversal. The algorithm is called H_1.

The following properties of Algorithm H_1 are easy to prove.

Theorem 7. *The output of Algorithm H_1 is planar.* ∎

Theorem 8. *All edges in the output of Algorithm H_1 are either horizontal or vertical.* ∎

Theorem 9. *Suppose that a binary tree $T = (V, E)$ of diameter $D \geq 2$ is drawn*



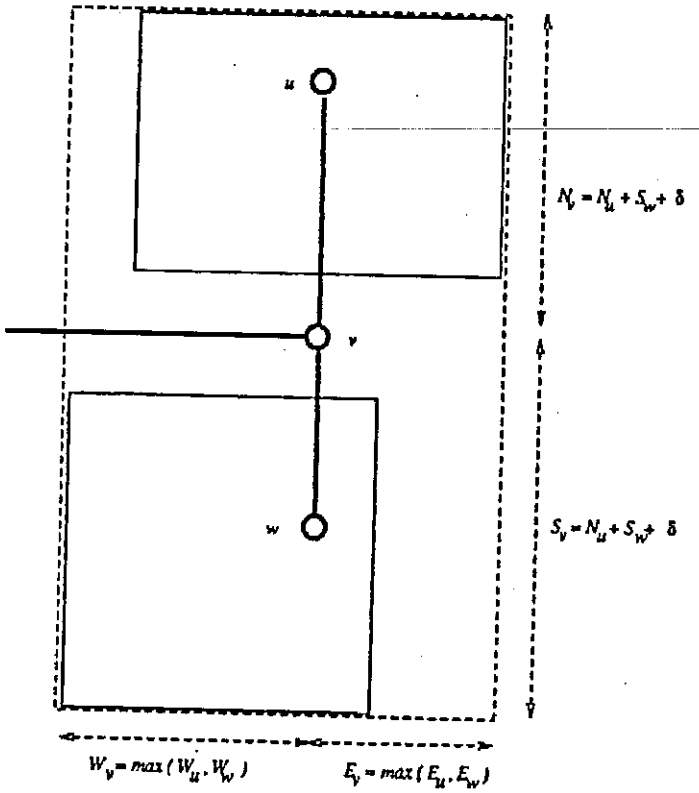


Figure 19: Computing subtree size from children's sizes

s drawn

by Algorithm H_1, and

$$L = \max_{(u,v)\in E} \|p(u)p(v)\|,$$
$$l = \min_{(u,v)\notin E} \|p(u) - (v)\|.$$

$N_v = N_u$

Then

$$L \leq lD$$

∎

Theorem 10. *Suppose that a tree of diameter D is drawn by Algorithm H_1. Define*

$S_v = S_u$

$$K = \max_{u,v\in V, u\neq v} \|p(u) - p(v)\|,$$
$$k = \min_{u,v\in V} \|p(u) - p(v)\|.$$

Then

$$k \geq \frac{1}{D}K.$$

∎

Theorem 11. *Suppose that v is a vertex of degree 2 in the output of Algorithm H_2 on T. Then the edges incident with v form a straight line.* ∎

This algorithm gives good results for binary trees. Sample results are in Figure 18.

6. Other Algorithms

Several other algorithms for drawing free trees are available. The VLSI literature gives algorithms for drawing free trees in linear area [19,25] and without bends in the edges [21]. Note, however, that the compactness of VLSI layout is not necessarily desirable for display, because the drawing can be very "tangled".

An interesting method is the "garden" layout. The longest path of the tree is drawn as a straight line along the bottom of the page, and the remainder of the tree "grows" upward from the path using rood tree drawing algorithms; this is illustrated in Figure 21. The diagrams may be compacted using a method similar to [18].

7. Remarks

All of the algorithms described in this paper, and many other tree drawing algorithms, have been implemented in the SPREMB graph drawing system [16]. Experience indicates that one can expect a good clear layout of a free tree with trees with 100 vertices on a canvas of size 150mm × 150mm, using circle of radius 1.2mm for each vertex.

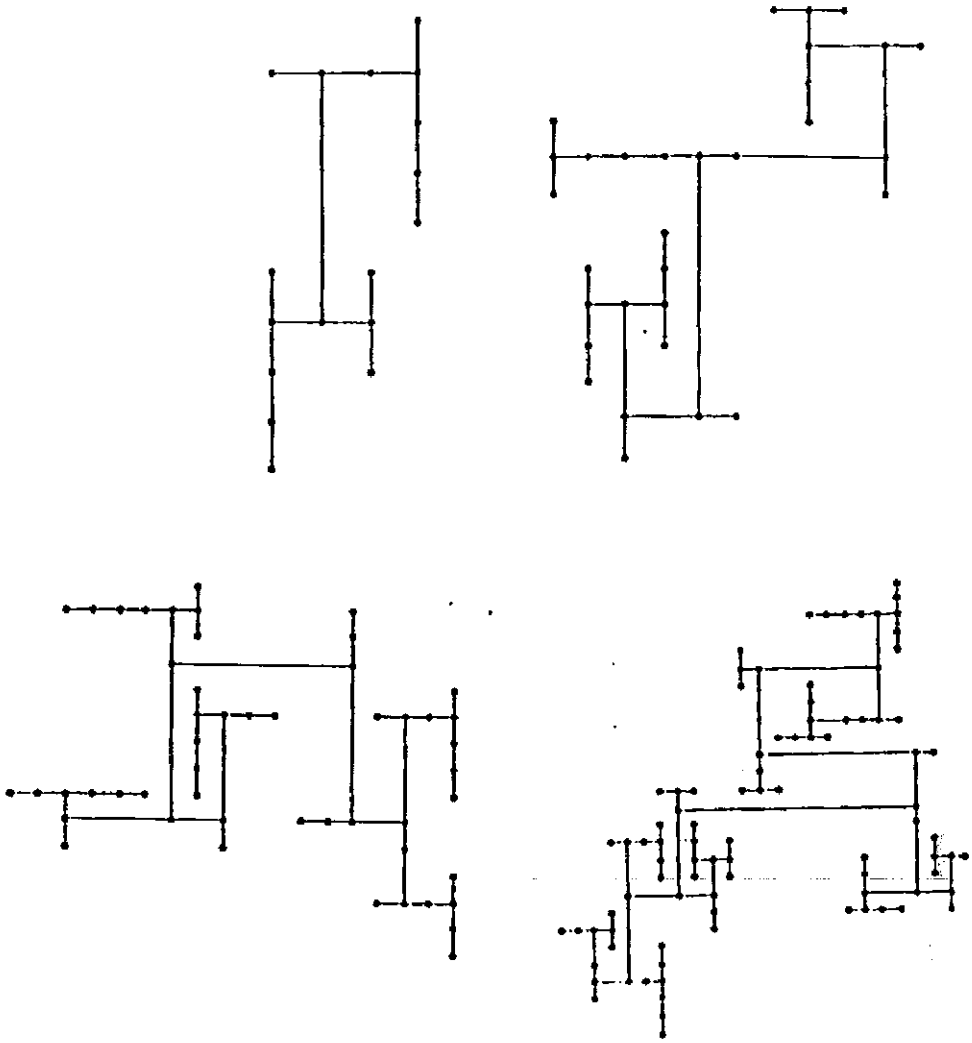


Figure 20: Sample output from Algorithm H_1

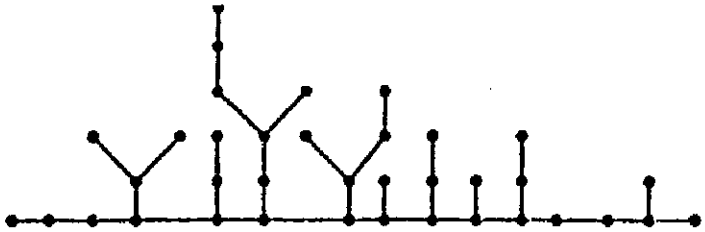


Figure 21: Example of a garden layout

Many of the algorithms have provably good properties, and it would be interesting to discover new algorithms which perform better with respect to variance in edge length and minimum distance between vertices. However, it is unlikely

that a universal tree drawing algorithm (which performs well for all trees) will be discovered. For a graph drawing system it seems more feasible to provide a number of algorithms and allow the user to choose a suitable layout.

Acknowledgement

The author is grateful for support from Fujitsu Laboratories, especially from Kozo Sugiyama and Kazuo Misue.

References

1. M.A Bernard, *On the automated drawing of graphs*, Proceedings of the Third Caribbean Conference on Combinatorics and Computing (1981), 43–55.
2. J.A. Bondy and U.S.R. Murty, "Graph Theory with Applications", MacMillan, 1977.
3. F.J. Brandenburg, *Nice drawing of graphs and trees are computationally hard.*, Technical Report MIP-8820, Fakultat fur Mathematik und Informatik, University of Passau (1988).
4. R. Davidson and D. Harel, *Drawing graphs nicely using simulated annealing.* (to appear).
5. P. Eades, *A heuristic for graph drawing*, Congressus Numerantium 42 (1984), 149–169.
6. P. Eades, X. Lin, and T. Lin, *Two tree drawing conventions*, in "Computational Geometry and its Applications", 1991. (to appear).
7. C. Esposito, *Graph graphics: Theory and practice*, Computers and Mathematics with Applications 15(4) (1988), 247–253.
8. T. Fruchterman and E. Reingold, *Graph drawing by force-directed placement*, Technical Report UIUCDCS-R-90-1609, UILU-ENG-90-1748, Department of Computer Science, University of Illinois at Urbana Champaign (1990). (to appear in Software Practice and Experience).
9. J. Hopcroft and R. Tarjan, *Isomorphism of planar graphs*, in "Complexity of Computer Computations", R. Miller and J. Thatcher, editors, Plenum, 1972.
10. T. Kamada, *Symmetric graph drawing by a spring algorithm and its application to radial drawing.* (to appear).
11. T. Kamada, *Visualizing Abstract Objects and Relations*, "Computer Science", Volume 5, World Scientific, Singapore, 1989.
12. T. Kamada and S. Kawai, *An algorithm for drawing general undirected graphs*, Information Processing Letters 31(1) (1989), 7–15.
13. D. Knuth, *Seminumerical Algorithms*, "The Art of Computer Programming", Volume 2, Addison Wesley, 1981.
14. X. Lin, *Springs and symmetry.* (to appear).
15. J. Manning and M.J. Atallah, *Fast detection and display of symmetry in trees*, Congressus Numerantium (1989).

16. I Fogg, P. Eades, and D. Kelly, *Spremb: a system for developing graph algorithms*, Congressus Numerantium **66** (1988), 123–140.
17. R.C. Read, *Methods for computer display and manipulation of graphs, and the corresponding algorithms*, Technical Report CORR 86-12, Faculty of Mathematics, University of Waterloo (1986).
18. E. Reingold and J. Tilford, *Tidier drawings of trees*, IEEE Transactions on Software Engineering **SE-7(2)** (1981), 223–228.
19. Y. Shiloach, *Arrangements of Planar Graphs on the Planar Lattice*, Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel (1976).
20. K. Supowit and E. Reingold, *The complexity of drawing trees nicely*, Acta Informatica **18** (1983), 377–392.
21. R. Tamassia, *On embedding a graph in the grid with a minimum number of bends*, SIAM Journal on Computing **16(3)** (1981), 421–444.
22. J.S. Tilford, *Tree drawing algorithms*, Master's thesis, Department of Computer Science, University of Illinois at Urbana Champaign (1981).
23. W.T. Tutte, *Convex representations of graphs*, Proc. London Math. Soc. **10(3)** (1960), 304–320.
24. W.T. Tutte, *How to draw a graph*, Proc. London Math. Soc. **13(3)** (1963), 743–768.
25. L. Valiant, *Universality considerations in VLSI circuits*, IEEE Transactions on Computers **C-30(2)** (1987), 135–140.
26. J. Vaucher, *Pretty printing of trees*, Software Practice and Experience **10(7)** (1980), 553–561.
27. C. Wetherall and A. Shannon, *Tidy drawings of trees*, IEEE Transactions on Software Engineering **SE-5(5)** (1979), 514–520.
28. S. Whitesides and R. Zhao, *On the placement of Euclidean trees*, Technical Report SOCS-89.03, School of Computer Science, McGill University (1989).