Optimal Layout of Edge-Weighted Forests*

Michael J. Fischer
Yale University

Michael S. Paterson
University of Warwick

August 25, 1998

Abstract

The layout problem for trees with weighted edges is motivated by the design of very large scale integrated circuits. Some of the nodes are fixed and the object is to position the remainder so that the total weighted edge cost is minimized. The cost of each edge is the product of its weight and its length under some appropriate norm. Optimization for planar layouts is shown to be NP-hard. If crossings are permitted, then optimal layouts under the L_1 norm can be efficiently computed. Suitable algorithms and data structures are presented, and explicit exact cost functions are given for two classes of weighted complete binary trees.

1 Introduction

We consider the problem of finding a minimal cost layout of a forest in Euclidean d-space. A *forest* is an acyclic undirected edge-weighted graph, and a *layout* is an assignment of a point in d-dimensional Euclidean space to each of the nodes of the forest. The "length" of an edge in the layout is the "distance" between its endpoints as measured by some norm. The *cost* of an edge is its length times its weight, and the *cost* of the whole layout is the sum of the costs of all the edges. We assume the positions of certain nodes are fixed in advance, and we wish to place the remaining nodes so as to minimize the cost of the layout.

Certain classes of forests are of particular interest. A *tree* is a connected forest. A tree is *binary* if all of its internal nodes have degree 3. A tree is *planted* if one of its leaves (i.e., degree 1 nodes) is distinguished as a root. A

*This work was supported by the National Science Foundation under Grant No. MCS77–02474, and by the EC under contract No. 7141 (project ALCOM II).

planted tree is *complete* if the number of edges on the path between any leaf and the root is the same. For example, Figure 1 shows a standard layout of the complete planted binary tree with 16 non-root leaves and unit edge weights, and Figure 2 shows a layout of the same tree which is optimal with respect to the total horizontal length of the edges, assuming that the edges run rectilinearly as shown and the leaves are fixed. (Think of these figures as representing one-dimensional layouts with the vertical lines inserted only for purposes of illustration.)

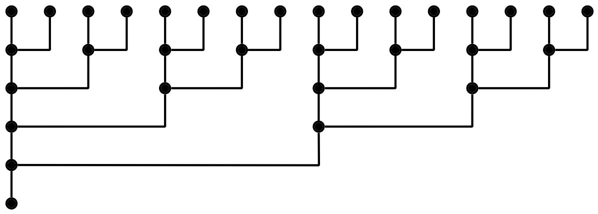

Figure 1: "Standard" layout with root at left edge.

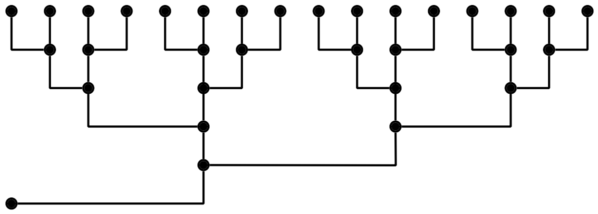

Figure 2: Optimal horizontal placement with root at left edge.

Convex functions play an essential role in our development. A function $f\colon \mathcal{R}^d \to \mathcal{R}$ is *convex* if

$$f(\gamma x + (1-\gamma)y) \leq \gamma f(x) + (1-\gamma) f(y)$$

for all $x, y \in \mathcal{R}^d$, $0 < \gamma < 1$. A convex function $\|\cdots\|$ is called a *norm* if it satisfies:

1. $\|x\| > 0 \Longleftrightarrow x \neq 0$;

2. $\|\gamma x\| = |\gamma| \cdot \|x\|$ for all $\gamma \in \mathcal{R}$.

We refer to two particular norms in this paper. The L_2-norm gives the usual Euclidean distance:

$$\|(x_1, \ldots, x_d)\|_{L_2} \stackrel{\text{df}}{=} \sqrt{\sum_t x_t^2}.$$

The L_1-norm is the "city block" metric:

$$\|(x_1, \ldots, x_d)\|_{L_1} \stackrel{\text{df}}{=} \sum_t |x_t|.$$

A *weighted forest* F is a pair (G, w), where $G = (V, E)$ is an undirected acyclic graph with vertex (node) set V and edge set E, and $w: E \to \mathcal{R}^+ \cup \{0\}$ is the edge weight function. (We write w_e for $w(e)$.) A *layout* of F is a function $\lambda: V \to \mathcal{R}^d$. We define the *length* of an edge $e = \{v, w\}$ in the layout λ (with respect to the norm $\|\cdots\|$) by

$$\text{length}(e) \stackrel{\text{df}}{=} \|\lambda(v) - \lambda(w)\|.$$

We define the cost of the layout by

$$\text{cost}_F(\lambda) \stackrel{\text{df}}{=} \sum_{e \in E} w_e \cdot \text{length}(e). \tag{1}$$

Let $V_L \subseteq V$ and $\chi: V_L \to \mathcal{R}^d$. The cost of the optimal layout with the placement of V_L fixed by χ is

$$C_F(\chi) \stackrel{\text{df}}{=} \min\{\text{cost}_F(\lambda) \mid \lambda: V \to \mathcal{R}^d \text{ and } \lambda \text{ extends } \chi\}. \tag{2}$$

Obviously, when χ fixes all nodes of F, we have $C_F(\chi) = \text{cost}_F(\chi)$.

The problem of finding optimal layouts with some nodes fixed is motivated by the problem of minimizing chip area in the design of very large scale integrated circuits [8]. Trees are commonly occurring sub-circuits, where the internal nodes are processing elements and the edges are wires connecting them. Each wire has some width and occupies an area equal to its length times its width. In some applications, different wires may have different widths, for they might have to carry varying amounts of current, or a wire might actually represent a bus consisting of many electrical pathways. We ignore the area occupied by the nodes on the (somewhat unrealistic) grounds that their total area is independent of their placement. Assuming fixed positions for the inputs and the output on the chip, the problem is to place the processing elements so as to minimize the area occupied by the wires.

Our abstract formulation of this problem makes the idealization that the nodes are points and can be placed arbitrarily close together (or can even coincide), although the constraints on the placement of the nodes in L will prevent the trivial layout with all nodes coincident. We also idealize wires to lines even though we talk about their "area"; thus the area occupied by overlapping wires is counted twice.

Brent and Kung [3] study the function $C_F(\chi)$ for F an unweighted complete planted tree in the plane, where χ fixes the non-root leaves. They show that if the leaves of the tree are constrained to be on the boundary of a convex region and at least unit distance apart, then the cost of the optimal layout is of order $n \log n$, and this is both an upper and a lower bound. In this paper, we focus on finding efficient algorithms for computing the cost and finding optimal layouts in some interesting special cases. A preliminary version of this paper appeared in [5].

2 Planar Layouts

We usually permit wires to cross, as in [3]. However, many applications in $\mathcal{R}^2$ require *planar layouts*, i.e., with no crossings. Here a layout must include the routings of the wires as well as the placement of the nodes. We let *planar*-$C_F(\chi)$ denote the greatest lower bound on the layout cost over all non-crossing wire routings and internal node placements, where the leaves are fixed by χ.[1] Unfortunately, computing *planar*-$C_F(\chi)$ is apparently an intractable problem for the L_2 and L_1 norms. We consider here the L_2 norm.

Let us define the PLANAR FOREST LAYOUT problem in the style of [6].

INSTANCE: Forest F, leaf positions χ, threshold k.

QUESTION: Is *planar*-$C_F(\chi) \leq k$?

Two special cases of PLANAR FOREST LAYOUT are of interest. In PLANAR TREE LAYOUT, the forest F consists of a single tree. In PLANAR STICKS LAYOUT, the forest F comprises a collection of disjoint edges or "sticks".

Theorem 1 *The PLANAR FOREST LAYOUT, PLANAR TREE LAYOUT, and PLANAR STICKS LAYOUT problems are all NP-hard, even when all edges have unit weight.*

[1] It would appear simpler to define *planar*-C_F in terms of the cost of the optimal layout. However, optimal layouts do not exist in some cases where wires may be placed arbitrarily close to nodes or other wires but must not touch.

Proof: It suffices to show that PLANAR TREE LAYOUT and PLANAR STICKS LAYOUT are NP-hard, since both are special cases of PLANAR FOREST LAYOUT.

We say a Boolean formula ϕ is in 3CNF if it is a conjunction of clauses, each of which contains at most three literals. Let ϕ be a 3CNF formula. Let $U = \{x_1, \ldots, x_n\}$ be the set of variables and $C = \{c_1, \ldots, c_m\}$ be the set of clauses that occur in ϕ. Associate with ϕ the graph $G_\phi = ((U \cup C, E_1 \cup E_2)$, where $E_1 = \{\{x_i, c_j\} \mid x_i$ or its negation appears in clause $c_j\}$, and $E_2 = \{\{x_i, x_{i+1}\} \mid 1 \leq i < n\} \cup \{\{x_n, x_1\}\}$. We define the PLANAR 3SAT problem (P3SAT), which is known to be NP-complete [6, 7].

INSTANCE: A 3CNF formula ϕ whose associated graph G_ϕ is planar.

QUESTION: Is ϕ satisfiable?

Our proof is completed by reducing P3SAT to each of PLANAR TREE LAYOUT and PLANAR STICKS LAYOUT. Given a planar layout of a 3CNF formula, we construct a planar sticks problem and a planar tree problem whose layout costs meet their respective targets if and only if the 3CNF formula is satisfiable. Both constructions are similar; indeed, the tree problem is obtained by adding edges to the sticks problem. Unfortunately, simply adding edges does not work in general, and we do not know of an easy reduction of PLANAR STICKS LAYOUT to PLANAR TREE LAYOUT.

In more detail, a planar layout of G_ϕ is converted to a planar sticks problem by replacing each variable, clause, and graph edge by one or more simple planar sticks designs, each consisting of a small number of pairs of points. Each design has a *target cost* and a number of distinct layouts having costs arbitrarily close to the target. The target cost for the whole planar sticks problem is the sum of the target costs for each design in it. The designs interact with each other, however, so the planar sticks problem will have a solution of the target cost if and only if the instance of P3SAT has a satisfying assignment.

It may be helpful in understanding our construction to consider a simple example. Let $\phi = (x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee w) \wedge (x \vee z \vee \neg w)$. A planar layout of the graph G_ϕ is shown in Figure 3. If some clause contains fewer than three literals, it is convenient to repeat an occurrence of one of its literals, adding a corresponding multiple edge to G_ϕ. We can assume then that each vertex in C has degree 3. Our construction replaces each edge in $E_1 \cup E_2$ with a "signal line" design, each vertex in U of degree $d \geq 3$ with a tree of $d-2$ copies of a "fanout" design, and each vertex in C with an appropriate "3SAT" design. A signal line is a long flexible design with special endpoints

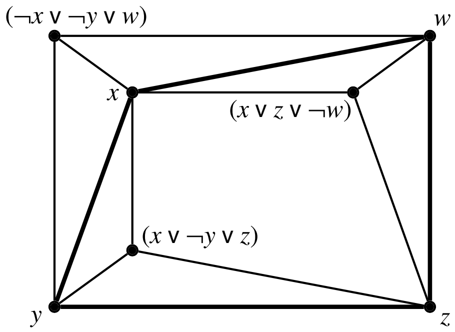

Figure 3: Graph of the formula ϕ.

used to join it up with other elements. Each endpoint has two target layouts that represent the two truth values. Signal lines come in three flavors. The first kind is like an electrical wire and is used to propagate a truth value from one end to the other. Its optimal layouts ensure that the truth values of the two ends are the same. The second kind is the same except that an implicit negation is built in. It ensures that the truth values of the two ends are different. We call these two kinds *active*. The third kind is *passive* and places no constraints on the two endpoints. The first two kinds of signal lines are used for edges from E_1, and the third kind is used for edges from E_2.[2] A fanout design attaches to three signal lines and achieves its target cost only when the attached endpoints of all three signal lines have the same truth value. A 3SAT design attaches to three signal lines and achieves its target cost only when at least one of the endpoints of the attached signal lines is true. It is used to represent a clause of the 3CNF formula.

We begin by considering a bistable element, shown in Figure 4. This is the basic building block of our constructions and is used for the endpoints of all signal lines. The lines (a, a') and (b, b') bisect each other at right-angles, and (a, a') is longer than (b, b'). The two "optimal" ways[3] of laying out the pairs are shown in Figure 5. These elements can be linked together to form a *signal line*, as shown in Figure 6. Note that this is fairly flexible and

[2]The edges in E_2 can be ignored when considering the PLANAR STICKS problem, but they are needed later when we connect up the sticks into a tree.

[3]Recall that $planar\text{-}C_F(\chi)$ is the greatest lower bound of the layout costs. Thus, it suffices for our construction to have layouts whose cost is arbitrarily close to the target cost, even if the target cannot actually be reached. We abuse terminology by calling such layouts *optimal*.

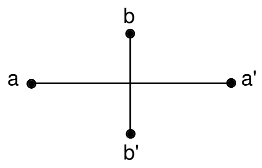

Figure 4: A bistable element.

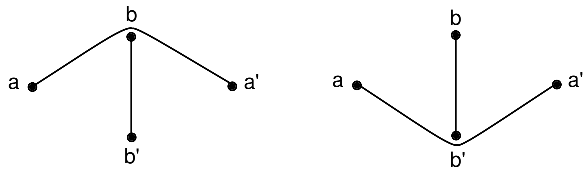

Figure 5: Two minimal configurations of a bistable element.

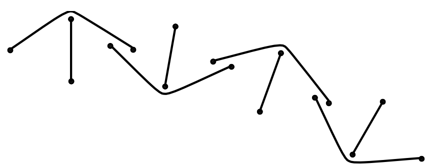

Figure 6: An active signal line.

the length is easily varied. Negation is effected by altering the parity of the number of bistable elements. A passive signal line is obtained by placing two active signal lines end-to-end but just failing to meet, as shown in Figure 7.

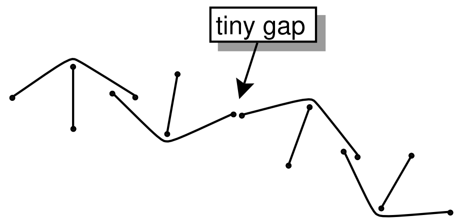

Figure 7: A passive signal line.

A *fanout* design is obtained from three bistable elements and three sticks, as shown in Figure 8. It has two distinct target-cost layouts: one is shown in Figure 9; the other is its mirror image. Simple geometric reasoning shows that the cost of any layout in which a line enters the central triangular region exceeds the target cost by more than some fixed amount. For example, Figure 10 shows the consequences of attempting to lay out one of the sticks in a straight line instead of wrapping it around the central triangle. This reduces the cost of the one stick but causes the third stick to take a circuitous route which more than compensates for the savings in laying out the first stick.

A *3SAT* design allows seven of the eight possible configurations of the three incoming signal lines and rejects the eighth. This corresponds to a clause with three literals and forces the clause to be satisfied. Our construction of a 3SAT element is based on three sticks that join opposite points of a regular hexagon and is shown in Figure 11.

A hexagon admits just two optimal layouts, up to rotational and reflectional symmetry, as shown in Figure 12. They both route one stick along a diameter of the hexagon, and they each have cost approaching $6+\sqrt{3} \approx 7.732$ (assuming a unit-side hexagon). The other minimal configuration (in the sense that each wire is optimally placed given the placement of the other two) is shown in Figure 13. Its cost exceeds $3 + 3\sqrt{3} \approx 8.196$, showing that it is not optimal. Thus, every optimal layout of the hexagon has exactly one stick placed along a diameter.

Referring now back to Figure 11, we see that every hexagon stick is incident on exactly one bistable element, and placing it on the diameter forces that element to a particular state. We now argue that, for each of

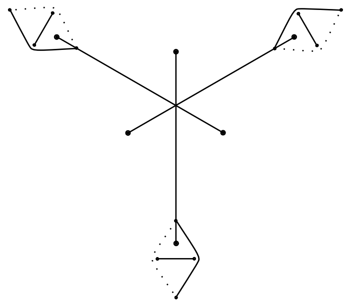

Figure 8: A fanout design.

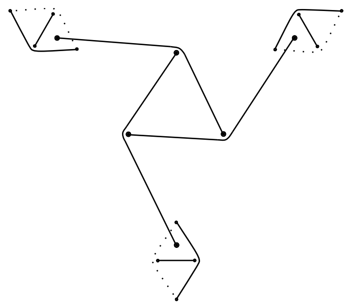

Figure 9: One of two optimal layouts of a fanout design.

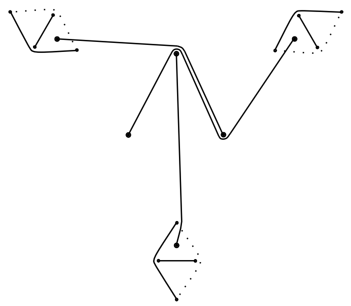

Figure 10: A non-optimal layout of a fanout design.

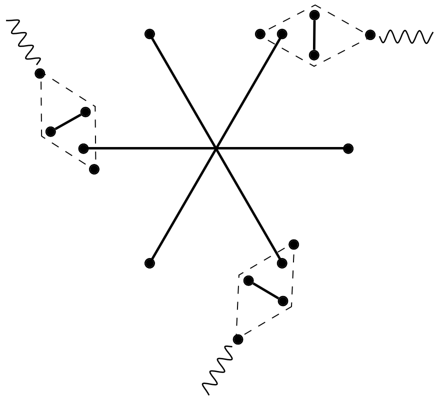

Figure 11: A 3SAT element.

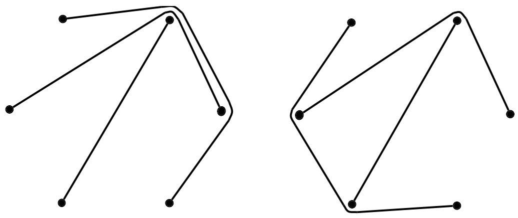

Figure 12: Optimal configurations of a hexagon.

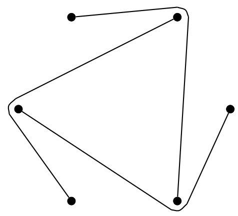

Figure 13: Minimal but non-optimal configuration of a hexagon.

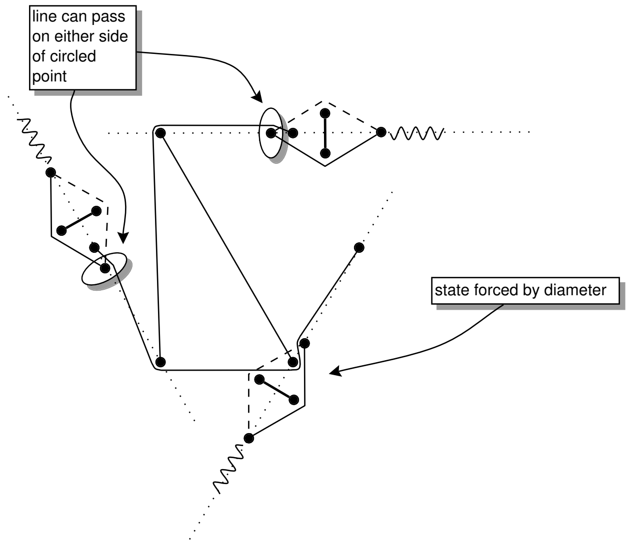

Figure 14: Optimal layout of a 3SAT element.

the four possible states of the other two bistable elements, the remaining hexagon sticks can still be placed so as to approach the optimal cost for the hexagon. Figure 14 shows one such case; the others are similar. Although the figure shows some lines wiggling to get around points, the dotted lines show that the points in question are collinear and hence the wiggly lines can be made as close to straight as desired.

Thus, the 3SAT element achieves its target cost except in the case that all three bistable elements are in the state with their stick routed towards the center of the hexagon. Assuming such a routing represents 0, a 3SAT element can be placed optimally if and only if the disjunction of its three inputs is satisfied.

This completes the construction of a planar sticks problem. It follows from the construction that the sticks can be laid out with cost approaching the target cost if and only if the original instance of P3SAT is satisfiable.

To obtain a planar tree problem, we connect the sticks together with additional edges. These joining edges are carefully chosen so that their optimal placements do not interfere with the intended placements of the sticks nor with each other. They simply add a constant to the target cost and to the cost of each sticks layout and otherwise have no effect. Thus, the planar tree problem will have solutions approaching their target cost if and only if the original instance of P3SAT has a satisfying assignment.

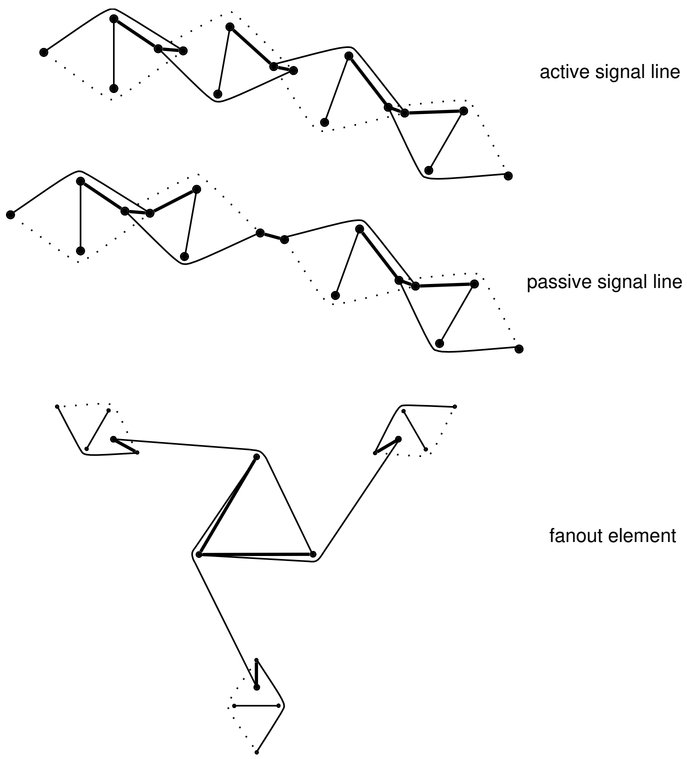

Figure 15: Connecting together signal lines and fanout elements.

Figure 15 shows how to connect the sticks together in active and passive signal lines and in a fanout element. The extra connecting edges are shown as heavy lines. Figure 16 show the extra connections for a 3SAT element. Here we connect each hexagon stick to one of the incident signal

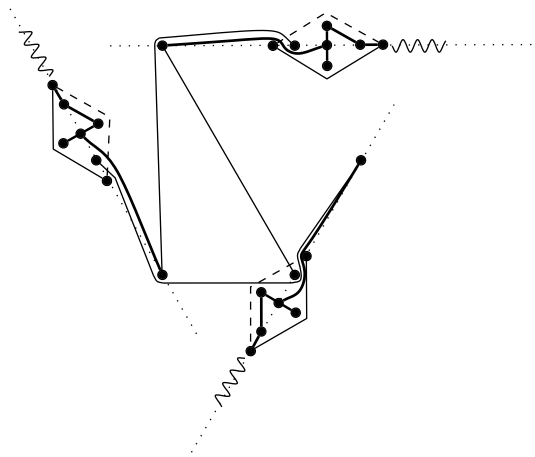

Figure 16: Connecting the hexagon sticks to the incident signal lines.

lines. Because the original P3SAT planar layout included a cycle connecting the variables together, the variables in our construction are also connected (via fanout elements and passive signal lines). Thus, the entire construction is connected. To make it into a tree, one need only remove one of the passive signal lines in the cycle connecting the variables. ∎

Becker [1] has shown that the planar tree layout problem is NP-hard over the L_1 metric also. The proof method is similar to that described in [5] but requires a number of new elements.

3 Properties of C_F

In this section, we explore the minimum cost layout function C_F. Without loss of generality, we can assume that F is a tree T and that the nodes whose positions are fixed in advance are just the leaves of T. If F contains more than one tree, or if an internal node of F is fixed, then the problem of laying out F reduces to the problem of independently laying out several smaller trees. If some leaf of T is not fixed, then an optimal layout places the leaf to coincide with its adjacent internal node, thereby reducing the length of its incident edge to zero. For a further obvious simplification we may assume that all internal nodes have degree at least three.

Regard T as a planted tree with non-root leaves $\ell_1, \ldots, \ell_n$, internal nodes $v_1, \ldots, v_n$, and root r. Let λ be the layout that places the nodes $\ell_1, \ldots, \ell_n$, $v_1, \ldots, v_n$, r at positions $x_1, \ldots, x_n$, $y_1, \ldots, y_m, z$, respectively. Let $\mathbf{x} = x_1, \ldots, x_n$ and $\mathbf{y} = y_1, \ldots, y_m$. We write $\mathrm{cost}_T(\mathbf{x}, \mathbf{y}, z)$ for $\mathrm{cost}_T(\lambda)$ and regard it as a function in $(\mathcal{R}^d)^{n+m+1} \to \mathcal{R}$. Similarly, let χ fix $\ell_1, \ldots, \ell_n$, r to $x_1, \ldots, x_n$, z, respectively. We write $C_T(\mathbf{x}, z)$ for $C_T(\chi)$ and regard it as a function in $(\mathcal{R}^d)^{n+1} \to \mathcal{R}$.

Theorem 2 *$\mathrm{cost}_T(\mathbf{x}, \mathbf{y}, z)$ and $C_T(\mathbf{x}, z)$ are convex functions.*

Proof: Let e be an edge in T. The quantity $\mathrm{length}(e)$ is a convex function of the positions of the endpoints of e. Since convex functions are closed under scalar multiplication and addition, it follows immediately from Equation (1) that cost_T is a convex function. By Equation (2), $C_T(\mathbf{x}, z) = \min_{\mathbf{y}} \mathrm{cost}(\mathbf{x}, \mathbf{y}, z)$. Since convex functions are closed under minimization, it follows that C_T is a convex function. ∎

The significance of this theorem is that general optimization methods may be applied to find the minimizing node positions, allowing one to compute C_T.

Computationally more efficient methods result from exploiting the structure of T. The equation

$$C_T(\mathbf{x}, z) = \min_y \left(\sum_i C_{T_i}(\mathbf{x}, y) + w \cdot \|y - z\| \right) \tag{3}$$

leads to an obvious dynamic programming algorithm to compute the minimum cost layout function for T in terms of the decomposition of T into maximal proper planted subtrees T_i, where w is the weight of the edge to T's root. This is illustrated in Figure 17.

The actual complexity of the algorithm will depend on finding a suitable representation for C_T which will permit efficient implementation of the necessary operations of addition and min. In the next section, we find such a representation for the L_1 metric, but the problem for the L_2 metric remains open.

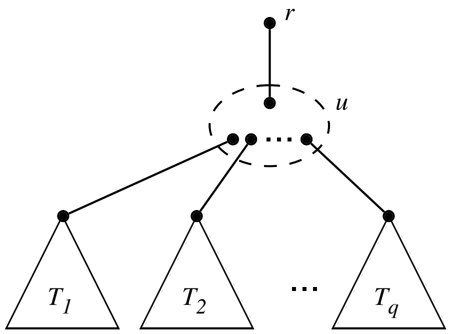

Figure 17: Decomposition of a planted tree.

4 C_T in the L_1 Metric

Applications of tree-layout, for example to chemical plant piping or certain kinds of circuit design, often demand "rectangular" layouts with all lines running parallel to one of the axes. This kind of distance is given by the L_1 metric. In the L_1 metric, a d-dimensional tree layout problem decomposes into d *independent* 1-dimensional problems, that is, one finds the optimal layout for each coordinate separately. We may thus restrict attention to just 1-dimensional problems, in which case $L_1 = L_2$.

The key to our optimal layout algorithm is a characterization of $C_T(\mathbf{x}, z)$ for fixed $\mathbf{x}$ and variable z. Throughout this section, we assume a fixed value for $\mathbf{x}$ and write $C_T(z)$ for $C_T(\mathbf{x}, z)$.

A function $f: \mathcal{R} \to \mathcal{R}$ is (finitely) *piecewise linear* if it is continuous and there are points $x_1 < \ldots < x_n$, called *joints*, such that $f(x)$ is linear in the neighborhood of x for all x not a joint. Let $f'^{-}(x)$ be the "left-hand" derivative of f at x (i.e., the limit of $(f(x') - f(x))/(x' - x)$ as x' approaches x from below), and let $f'^{+}(x)$ be the "right-hand" derivative of f at x (i.e., the limit of $(f(x') - f(x))/(x' - x)$ as x' approaches x from above). For convenience of notation, let $x_0 = -\infty$, $x_{n+1} = +\infty$, and extend f'^{-} and f'^{+} by defining $f'^{+}(x_0) = f'^{-}(x_1)$ and $f'^{-}(x_{n+1}) = f'^{+}(x_n)$. Both f'^{-} and f'^{+} are step functions with discontinuities only at the joints, and $f'^{+}(x_i) = f'^{-}(x_{i+1})$ for $0 \leq i \leq n$.

If a piecewise linear function $f(x)$ is convex, then $f'^{-}(x) \leq f'^{+}(x)$ for all x, with equality holding except at the joints, and $f'^{-}(x)$ and $f'^{+}(x)$ are non-decreasing. It is easily seen that the sum of convex piecewise linear functions is again convex piecewise linear, and its set of joints is the union of the set of joints of the two summands. Observe that $f(x)$ is minimized at a point $x_{\min}$ such that $f'^-(x_{\min}) \leq 0$ and $f'^+(x_{\min}) \geq 0$. Any such point is a joint unless $f'^-(x_{\min}) = f'^+(x_{\min}) = 0$.

Theorem 3 *For any planted tree T and fixed $x_1 \leq \ldots \leq x_n$, $C_T(z)$ is non-negative convex piecewise linear with joints only at $x_1, \ldots, x_n$. Moreover, if w is the weight of the root edge of T, then $-{C'_T}^+(x_0) = {C'_T}^-(x_{n+1}) \leq w$, where $x_0 = -\infty$ and $x_{n+1} = +\infty$.*

Proof: By induction on n, the number of non-root leaves of T.

If $n = 1$ and w is the weight of the edge from the root to the one leaf of T, the optimal layout is a single wire from root to leaf. It has cost $w \cdot |z - x_1| \geq 0$, which is non-negative convex piecewise linear with a joint at x_1 (if $w > 0$). Moreover, ${C'_T}^+(x_0) = -w$ and ${C'_T}^-(x_2) = w$, completing this case.

Now suppose $n > 1$. Then T can be decomposed into planted subtrees $T_1, \ldots, T_q$ ($q \geq 2$) as shown in Figure 17. We fix the position of the root r at z, node u at y and then optimally lay out each of the trees $T_1, \ldots, T_q$. The cost of the resulting layout is

$$g_z(y) = h(y) + w \cdot |z - y|, \tag{4}$$

where

$$h(y) = \sum_{i=1}^{q} C_{T_i}(y). \tag{5}$$

The optimal cost for T is then

$$C_T(z) = \min_y g_z(y). \tag{6}$$

We must show that C_T satisfies the conditions of the theorem.

By induction, each $C_{T_i}(y)$ is non-negative convex piecewise linear with joints only at the x_j's corresponding to leaves that occur in T_i. Hence $h(y)$ is non-negative convex piecewise linear with joints only at the x_j's. Moreover, $-h'^+(x_0) = -\sum_i {C'_{T_i}}^+(x_0) = \sum_i {C'_{T_i}}^-(x_{n+1}) = h'^-(x_{n+1})$.

Define a, b, for $0 \leq a \leq b \leq n+1$, by

$$\left.\begin{aligned} a &= \min\{\, i \mid -w < h'^+(x_i) \,\} \\ \text{and} \quad b &= \max\{\, i \mid h'^-(x_i) < w \,\} \end{aligned}\right\} \tag{7}$$

It follows that

$$h'^-(x_a) \leq -w < h'^+(x_a) \tag{8}$$

and

$$h'^{-}(x_b) < w \leq h'^{+}(x_b). \tag{9}$$

We will show that these points x_a and x_b determine an optimal placement of node u. When $z \in [x_a, x_b]$, the optimal position of u is coincident with the root, and no root wire is used. When z lies outside of this interval, u is optimally placed at the endpoint of the interval nearest z, and a root wire connects u with r.

The value of y that minimizes $g_z(y)$ and the values of $C_T(z)$, ${C'_T}^{-}(z)$, and ${C'_T}^{+}(z)$ are given in Table 1. The proof for the minimizing y in all cases

Table 1: Intervals of interest in computing $C_T(z)$.

Range of z	y	$C_T(z)$	${C'_T}^{-}(z)$	${C'_T}^{+}(z)$
$(-\infty, x_a)$	x_a	$h(x_a) + w \cdot \lvert z - x_a \rvert$	$-w$	$-w$
x_a	z	$h(z)$	$-w$	$h'^{+}(z)$
(x_a, x_b)	z	$h(z)$	$h'^{-}(z)$	$h'^{+}(z)$
x_b	z	$h(z)$	$h'^{-}(z)$	w
(x_b, ∞)	x_b	$h(x_b) + w \cdot \lvert z - x_b \rvert$	w	w

involves showing that ${g'_z}^{-}(y) \leq 0$ and ${g'_z}^{+}(y) \geq 0$. It uses the facts that $g_z(y) = h(y) + w \cdot |y - z|$ together with obvious facts about the left and right derivatives of absolute value. We leave the details to the reader. Once we have established the minimizing y, constructing the remainder of the table is straightforward. The fact that $C_T(z)$ is non-negative convex piecewise linear with joints only at $x_1, \ldots, x_n$ follows from the table and the same fact for $h(z)$. From the table, it follows that $-{C'_T}^{-}(x_1) = {C'_T}^{+}(x_n) = w$ as desired. ∎

Theorem 3 shows that $n + 2$ numbers suffice to represent the function C_T, namely, the $n + 1$ slopes $s_i = {C'_T}^{+}(x_i) = {C'_T}^{-}(x_{i+1})$, $i = 0, 1, \ldots, n$, and the value of $C_T(0)$. Given that representation, $C_T(z)$ can be computed in time $O(n)$ by integrating from 0 to z and adding in the value of $C_T(0)$. The representation itself can be computed by an obvious recursive algorithm based on the decomposition of Figure 17, Equation 6, and Table 1. The time of that algorithm, $\mathrm{Time}_T(n)$, satisfies the recurrence equation

$$\mathrm{Time}_T(n) \leq \sum_{i=1}^{q} \mathrm{Time}_{T_i}(n_i) + O(n)$$

where n_i is the number of leaves in subtree T_i. If T is complete and of bounded degree, then $\mathrm{Time}_T(n) = O(n \log n)$, but in general we have only $\mathrm{Time}_T(n) = O(n^2)$, and when T is a chain of depth $n-1$ that bound is tight. We shall now present a more efficient algorithm which runs in $O(n \log n)$ time no matter how unbalanced the tree.

5 Efficient Algorithms

We may represent functions ${C'_T}^-$ and ${C'_T}^+$ by the extreme values $s^- = s_0$ and $s^+ = s_n$, and the "steps" $\alpha_i = s_i - s_{i-1}$ at each x_i. For any tree we will compute the pair (s^-, D), where D is the set of pairs $\{(i, \alpha_i) : 1 \leq i \leq n\}$, recursively from the pairs corresponding to the immediate subtrees. (We do not need to explicitly compute s^+ since $s^+ = -s^-$.) After an outline of the processes of the algorithm we will discuss data structures appropriate to an efficient implementation.

Corresponding to Equation (5), in the recursive evaluation, we merely sum the extreme values and take the union of the sets D for the q subtrees. Let the resulting steps be denoted $\overline{\alpha}_i$ and the corresponding slopes by $\overline{s}_i$. Now we must compute the new values of α_i derived from Equations (4) and (6).

Define a, b as in Equation (7) in the proof of Theorem 3. It follows from Table 1 that a minimizing value of y for Equation (4) is given as a function of the root position z by

$$y = \begin{cases} x_a & \text{if } z < x_a \\ z & \text{if } x_a \leq z \leq x_b \\ x_b & \text{if } x_b < z \end{cases}$$

and the slopes are given by

$$s_k = \begin{cases} -w & \text{if } k < a \\ \overline{s}_k & \text{if } a \leq k < b \\ w & \text{if } k \geq b\ . \end{cases} \tag{10}$$

Hence,

$$\alpha_k = \begin{cases} 0 & \text{if } k < a \\ \overline{s}_a + w & \text{if } k = a \\ \overline{\alpha}_k & \text{if } a < k < b \\ w - \overline{s}_{b-1} & \text{if } k = b \\ 0 & \text{if } k > b\ . \end{cases} \tag{11}$$

We can compute the $\overline{s}_i$'s by either of the following formulae:

$$\left.\begin{array}{c} \overline{s}_i = \overline{s}_0 + \sum_{j=1}^{i} \overline{\alpha}_j \\ \text{or} \\ \overline{s}_i = \overline{s}_n - \sum_{j=i+1}^{n} \overline{\alpha}_j \end{array}\right\} \quad (12)$$

We observe from (8) and (9) that, for $k = a$ or b, $1 \le k \le n$, we have $\overline{\alpha}_k > 0$. It follows from (11) for all i that if $\overline{\alpha}_i = 0$ then $\alpha_i = 0$. Therefore at any leaf in a subtree where $\alpha_i = 0$, this value will never change as a result of combining the subtree with others.

Our algorithm will be based on (7) and (10)–(12), so it is evident that the computation of subsequent stages would be unaffected by the removal from consideration of any leaf at which $\alpha = 0$. To improve the efficiency of our algorithm we therefore delete such a leaf from the active part of our data structure. The active set of steps is held as a sequence V, ordered by leaf position. We assume five basic operations on sequences:

- merge$(V_1, \ldots, V_q)$ returns an ordered sequence containing the union of the elements of $V_1, \ldots, V_q$;
- first(V) returns a reference to a pair (i, α) in V with i minimal;
- last(V) returns a reference to a pair (i, α) in V with i maximal;
- delete_first(V) modifies V by removing the pair first(V);
- delete_last(V) modifies V by removing the pair last(V).

Algorithm A

{Given a tree T with n leaves fixed to positions $x_1 < x_2 < \ldots x_n$ respectively, produce the data structure (s^-, V) representing the slopes of C_T, and output the a and b values corresponding to T as a_T and b_T.}

If T has a single leaf i with root edge weight $w > 0$ then

$$s^- = -w, \; V = ((i, 2w)), \; a_T = b_T = i.$$

If T has a single leaf i with root edge weight $w = 0$ then

$$s^- = 0, \; V = (), \; a_T = b_T = i.$$

If T has root edge weight w and subtrees $T_1, \ldots, T_q$, then apply the algorithm recursively to each subtree T_r to obtain (s_r^-, V_r).

$s^- := \sum_{r=1}^{q} s_r^-$
$s^+ := -s^-$
$a_T := 0$
$b_T := n+1$
{where $x_0 = -\infty,\ x_{n+1} = +\infty$ }
$V :=$ merge$(V_1, \ldots, V_q)$
If $s^- \leq -w$ then begin
    $(i, \overline{\alpha}) :=$ first(V)
    $S := s^- + \overline{\alpha}$
    while $S \leq -w$ do begin
        delete_first(V)
        $(i, \overline{\alpha}) :=$ first(V)
        $S := S + \overline{\alpha}$
        {comment: $S = \overline{s}_i$ }
    end
    $a_T := i$
    first$(V) := (i, S + w)$
    $(i, \overline{\alpha}) :=$ last(V)
    $S := s^+ - \overline{\alpha}$
    while $S \geq w$ do begin
        delete_last(V)
        $(i, \overline{\alpha}) :=$ last(V)
        $S := S - \overline{\alpha}$
        { comment: $S = \overline{s}_{i-1}$ }
    end
    $b_T := i$
    last$(V) := (i, w - S)$
    $s^- := -w$
end
return (s^-, V), a_T, b_T

■

From the results of Algorithm A for a tree T we can compute $\mathbf{s}$ in linear time by the following algorithm.

Algorithm B

{Given the data structure (s, V) produced by Algorithm A acting on a tree T, compute $\mathbf{s}$.}

> Initialize array $\hat{\alpha}$ to 0
> Remove each pair (i, α) in any order from V and do
> $\hat{\alpha}_i := \alpha$
> $s_0 := s$
> for $i := 1$ to n do begin
> $s_i := s_{i-1} + \hat{\alpha}_i$
> end

■

Theorem 4 *Given a planted tree T of N nodes with leaf positions fixed to $\mathbf{x}$, the slope vector $\mathbf{s}$ and the a, b values for T and each of its subtrees can be computed in time $O(n \log n)$. $C_T(\mathbf{x}, z)$ and an optimal layout for T can then be computed in additional time $O(N)$ for any root position.*

Proof: We shall analyze the running time of Algorithms A and B in terms of the numbers of basic operations used. It is clear from the form of these algorithms that these numbers are at most linear in N. The number of calls of Algorithm A is $N - 1$.

Apart from our five special operations on sequences, each instruction takes a constant time. We shall describe below data structures with the following complexity characteristics.

Operation	Time
delete_first(V)	$O(\log \lvert V \rvert)$
delete_last(V)	$O(\log \lvert V \rvert)$
first(V)	$O(\log \lvert V \rvert)$
last(V)	$O(\log \lvert V \rvert)$
merge($V_1, \ldots, V_q$)	$O\left(\sum_{r=1}^{q} \mu_r \log \frac{\mu_0}{\mu_r}\right)$

where $\mu_r = |V_r|$ and $\mu_0 = \sum_{r=1}^{q} \mu_r$.

To satisfy our claim to an $O(N \log N)$ time algorithm, it is now sufficient to consider the total time spent on merge operations. Suppose tree

T_0 is composed of q immediate subtrees $T_1, \ldots, T_q$, and $V_0, V_1, \ldots, V_q$ are the corresponding sequences generated by Algorithm A. V_0 is obtained by merging $V_1, \ldots, V_q$ and then deleting some of the elements. Suppose T_r has m_r leaves, then $\mu_r = |V_r| \leq m_r$ for $0 \leq r \leq q$.

Lemma 1 *Total merge time for a tree T with m leaves is $O(m \log m)$.*

Proof: We show by induction that the total merge time is at most $km \log m$ for some k.

The lemma is true for $m = 1$. Suppose T_0 has m_0 leaves and the lemma holds for $m < m_0$. Assume T_0 has subtrees $T_1, \ldots, T_q$ with $q \geq 2$, and that the total merge time to compute $V_1, \ldots, V_q$ is at most

$$k \sum_{r=1}^{q} m_r \log m_r \ .$$

For sufficiently large k, the total merge time for T_0 is at most

$$\begin{aligned} & k \left(\sum_{r=1}^{q} m_r \log m_r \right) + k \left(\sum_{r=1}^{q} \mu_r \log \frac{\mu_0}{\mu_r} \right) \\ \leq \quad & k \sum_{r=1}^{q} \left(m_r \log m_r + m_r \log \frac{m_0}{m_r} \right) \qquad \text{(by monotonicity in each } \mu_r) \\ = \quad & k m_0 \log m_0 \ . \end{aligned}$$

This completes the inductive proof. ■

A data structure which achieves these bounds is the 2–3 tree with "fingers" which is described by Brown and Tarjan [4]. Even without "fingers" the accessing and deleting operations take only $O(\log N)$ time each. In §5 of [4] the authors establish a bound of $O(s + s\log(t/s)) = O(s \log((t+s)/s))$ for the time to merge two lists of size s, t, with $s \leq t$. Our q-way merge can be accomplished by successively merging the $q-1$ smallest lists into the largest, V_1 say, for a total time of

$$O\left(\sum_{r=1}^{q} \mu_r \log \frac{\sum_{j \leq r} \mu_j}{\mu_r} \right) \leq O\left(\sum_{r=1}^{q} \mu_r \log \frac{\mu_0}{\mu_r} \right)$$

as required.

While the computation of $C_T(\mathbf{x}, z)$ in linear time from $\mathbf{s}$ is obvious, the derivation of an optimal layout requires some explanation.

An optimal location for the internal node nearest the root of a tree T is given as a function of the root position, z, by

$$y = \begin{cases} x_a & \text{if } z < x_a \\ z & \text{if } x_a \leq z \leq x_b \\ x_b & \text{if } x_b < z \end{cases}$$

and we have calculated the indices a, b for every tree in Algorithm A. The location of each internal node in an optimal layout, as a function of z, can be computed by a recursive procedure working from the root down.

Suppose we have computed this location function for an internal node v and it has the form

$$y_v = \text{Band}(z, \ell_v, r_v)$$

where

$$\text{Band}(z, j, k) \stackrel{\text{df}}{=} \begin{cases} x_j & \text{if } x < x_j \\ z & \text{if } x_j \leq z \leq x_k \\ x_k & \text{if } x_k < z. \end{cases}$$

To locate an internal node u immediately below v, consider the subtree T_u containing u, rooted at v. Given the computed indices a_u, b_u for T_u, we have

$$\begin{aligned} y_u &= \text{Band}(y_v, a_u, b_u) \\ &= \text{Band}(\text{Band}(z, \ell_v, r_v), a_u, b_u) \\ &= \text{Band}(z, \ell_u, r_u) \end{aligned}$$

where

$$\ell_u = \begin{cases} a_u & \text{if } \ell_v < a_u \\ \ell_v & \text{if } a_u \leq \ell_v \leq b_u \\ b_u & \text{if } b_u < \ell_v \end{cases}$$

and

$$r_u = \begin{cases} a_u & \text{if } r_v < a_u \\ r_v & \text{if } a_u \leq r_v \leq b_u \\ b_u & \text{if } b_u < r_v\ . \end{cases}$$

Thus u also has a location function of the assumed form. The complete layout as a function of z is computable in linear time. ■

There are two special cases of interest, in both of which the time for merging is reduced and the cost vectors can be generated in only $O(N)$ time.

Theorem 5 *If all of the edge weights are equal (the* unit cost *case) then* **s** *and* $C_T(\mathbf{x}, z)$ *are computable in* $O(N)$ *time.*

Proof: If all weights are unity then the computation is considerably simplified. The gradient of C_T with respect to z is always -1, 0, or 1 and so C_T has the form of Figure 18, where x_a, x_b may possibly coincide.

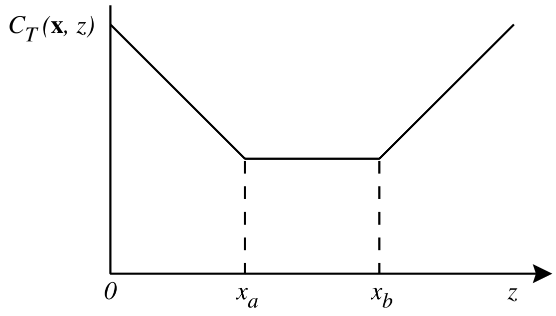

Figure 18: $C_T(\mathbf{x}, z)$ as a function of root position z in the unit cost case.

In terms of the data structure, we have $s = -1$ and $\alpha_i = 0$, except when $i \in \{a, b\}$ in which case $\alpha_a = \alpha_b = 1$ (or $\alpha_a = 2$ when $a = b$). These properties are easy to prove inductively. The consequences are that for any tree the active list V is of length one or two and has the form either $((a, 1), (b, 1))$, or $((a, 2))$ when $a = b$. If q, the number of immediate subtrees, is bounded by a constant then the time for the merge operation is similarly bounded and the total time for the algorithm we have already described is linear in the number of nodes.

If q is unbounded, growing with N, then a modification to the algorithm is needed to achieve linearity. We find that $s^- = -q$ and $s^+ = +q$, and each α_i, $i = 1, \ldots, q$, contributes $+1$ to the gradient $\overline{s}_i$. The indices a, b, marking the ends of the (possibly empty) interval in which $\overline{s} = 0$, are therefore given by the two median elements of the multiset $\{a_r \mid 1 \leq r \leq q\} \uplus \{b_r \mid 1 \leq r \leq q\}$. Given a and b, the list elements corresponding to all other indices are removed, leaving the new small list V. Since medians can be found in linear time [2], the total time required can be shown to be linear in the number of nodes of T. ∎

An ordering of the leaf positions of a tree is *natural* if the leaves of any pair of disjoint subtrees lie in two disjoint intervals.

Theorem 6 *If the given ordering on the leaves is a natural ordering then* **s** *and* C_T *are computable in* $O(N)$ *time.*

Proof: When the cost vectors are computed recursively in the order induced by the leaf ordering, it is found that the active lists V require no merging, they may just be concatenated. Therefore, a more efficient data structure for V is a doubly linked list. It now costs just a constant time to concatenate two lists and the total time is again found to be $O(N)$. ■

We make one final remark about the function C_T when T is a planted binary tree with unit edge weights and leaf positions in natural order. In this case, the joints of Figure 18 occur at the positions of the leaves $\ell_{1/3}$ and $\ell_{2/3}$ which are defined by the "1/3, 2/3" rule:

> $\ell_{1/3}$ is the leaf reached by starting at the root, taking the left branch, then the right, and continuing alternately left and right until encountering a leaf. $\ell_{2/3}$ is defined similarly but beginning with the right branch.

Using this rule, an optimal layout is easily constructed without actually computing the cost function, for the node u directly under the root of T is placed to coincide with the root if the root lies between $\ell_{1/3}$ and $\ell_{2/3}$, and otherwise it is placed to coincide with the one of those two leaves nearest the root. The rest of the tree is laid out using this rule recursively on the two subtrees.

6 Closed-Form Expressions for C_T in Special Cases

In this section, we assume T is a complete planted binary tree with $n = 2^\nu$ non-root leaves and a root r. For $1 \leq i \leq n$, let $x_i = i$, and assume $\mathbf{x}$ is in natural order for T. Let z be the position of r. We find explicit, closed-form descriptions of $C_T(\mathbf{x}, z)$ for two different edge weight assignments: *unit cost*, where $w_e = 1$ for all edges e, and *logarithmic cost*, where w_e is the "height" of e, that is, $w_e = 1 + \log_2(\text{number of leaves in the subtree defined by } e)$. Thus, $w_e = 1$ if u is a leaf, and $w_e = \nu + 1$ if $v = r$.

Theorem 7 *For T and $\mathbf{x}$ restricted as above and unit cost edge weights,*

$$C_T(\mathbf{x}, z) = \begin{cases} c(\nu) + a(\nu) - z & \textit{if } z < a(\nu); \\ c(\nu) & \textit{if } a(\nu) \leq z \leq b(\nu); \\ c(\nu) + z - b(\nu) & \textit{if } z > b(\nu); \end{cases}$$

where

$$c(\nu) \quad = \quad \frac{1}{9} \cdot ((3\nu + 1) \cdot 2^\nu - (-1)^\nu) \sim \frac{1}{3} \cdot n \log n$$

$$\begin{aligned} a(\nu) &= \frac{1}{3}\cdot 2^\nu + \frac{1}{2} + \frac{1}{6}\cdot(-1)^\nu \sim \frac{1}{3}\cdot n \\ b(\nu) &= \frac{2}{3}\cdot 2^\nu + \frac{1}{2} - \frac{1}{6}\cdot(-1)^\nu \sim \frac{2}{3}\cdot n \end{aligned}$$

Proof: From the recursive construction of C_T from C_{T_1} and C_{T_2} for its two (identical) subtrees T_1 and T_2, we get:

$$a(0) = b(0) = 1,\ c(0) = 0$$

and for $\nu \geq 1$,

$$\begin{aligned} a(\nu) &= b(\nu-1) \\ b(\nu) &= a(\nu-1) + 2^{\nu-1} \\ c(\nu) &= 2\cdot c(\nu-1) + a(\nu-1) - b(\nu-1) + 2^{\nu-1} \end{aligned}$$

It is easily verified that the given expressions for c, a, and b satisfy these recurrences. ■

This cost of $(1/3)n\log n$ compares to the cost of $(1/2)n\log n$ obtained by placing each node midway between its two sons as in Figure 19.

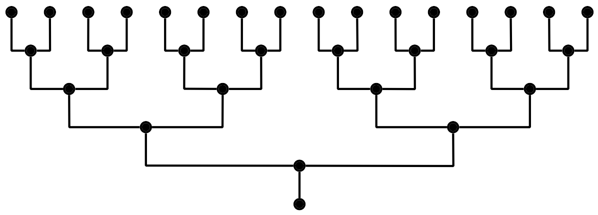

Figure 19: Symmetric tree layout.

Theorem 8 *For T and $\mathbf{x}$ restricted as above and logarithmic cost edge weights,*

$$\min_z C_T(\mathbf{x}, z) = c_B(\nu) = \left(\frac{1}{6}\,\nu^2 + \frac{7}{90}\,\nu + \frac{106}{675}\right)\cdot 2^\nu + \beta(\nu)$$

where β is the periodic function with period 4 given by the table:

$\nu \bmod 4$	$\beta(\nu)$
0	$-106/675$
1	$133/675$
2	$56/675$
3	$-83/675$

Proof: In this case, $C_T(\mathbf{x}, z)$ has the form illustrated in Figure 20. The slope is $-(\nu+1)$ to the left of X_A, 0 between X_B and X_C, and $(\nu+1)$ to the right of X_D. In between X_A and X_B, the slope is increasing and integer-valued, and is bounded by the interval $[-\nu, -1]$. In between X_C and X_D, the slope is again increasing and integer-valued, and it is bounded by the interval $[1, \nu]$.

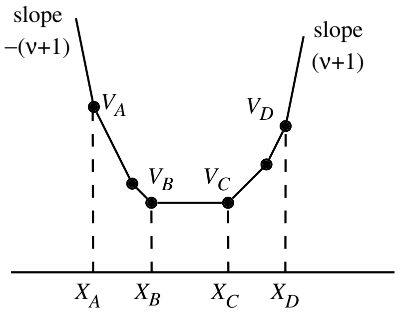

Figure 20: $C_T(\mathbf{x}, z)$ as a function of z for logarithmic costs.

Let V_A, V_B, V_C, V_D be the values of $C_T(\mathbf{x}, z)$ for $z = X_A, X_B, X_C, X_D$, respectively. These four values can be computed recursively even without obtaining a complete representation for C_T. Let T' and T'' be the two maximal proper planted subtrees of T, and let X'_i, V'_i and X''_i, V''_i be the corresponding points and values $i \in \{A, B, C, D\}$. Then X_i, V_i can be expressed in terms of these quantities as illustrated in Figure 21. We obtain the following recurrences:

$$\left.\begin{array}{l} X_A = X'_B,\ X_B = X'_D,\ X_C = X''_A,\ X_D = X''_C \\ V_A = V'_B + V''_A + \nu \cdot (X''_A - X'_B) \\ V_B = V'_D + V''_A + \nu \cdot (X''_A - X'_D) \\ V_C = V_B \\ V_D = V''_C + V'_D + \nu \cdot (X''_C - X'_D) \end{array}\right\} \tag{13}$$

These equations are in fact correct for arbitrary planted binary trees with leaves in natural order. However, when T is a complete planted binary tree with 2^ν leaves placed at the points $1, \ldots, 2^\nu$, then T' and T'' are isomorphic, and C_T itself is symmetric about the midpoint between X_B and X_C. Letting $X_i(\nu)$ and $V_i(\nu)$ denote X_i and V_i respectively, $i \in \{A, B, C, D\}$,

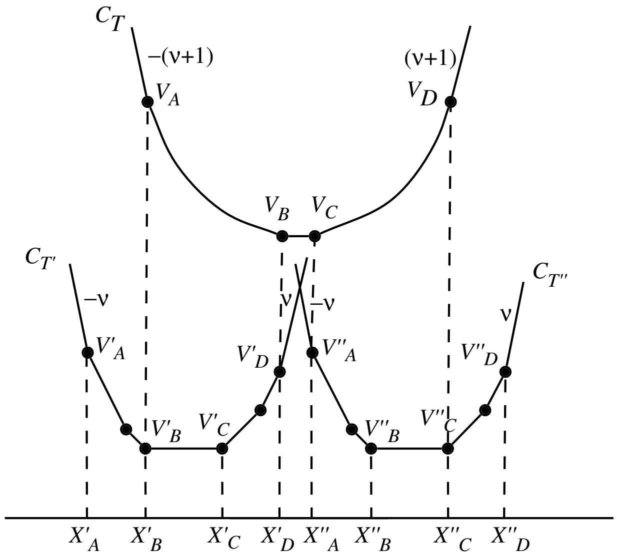

Figure 21: Recursive construction of C_T for logarithmic cost trees.

Equations (13) simplify to (14) and (15):

$$\left.\begin{array}{l} V_A(\nu) = V_A(\nu-1) + V_B(\nu-1) + \nu \cdot (p(\nu) + q(\nu))/2 \\ V_B(\nu) = 2 \cdot V_A(\nu-1) + \nu \cdot q(\nu) \end{array}\right\} \tag{14}$$

where $p(\nu) = X_D - X_A$ and $q(\nu) = X_C - X_B$ satisfy

$$\left.\begin{array}{l} p(\nu) = 2^{\nu-1} + q(\nu-1) \\ q(\nu) = 2^{\nu-1} - p(\nu-1) \end{array}\right\} \tag{15}$$

Also, $p(0) = q(0) = V_A(0) = V_B(0)$. The solution to these recurrences is:

$$\begin{aligned} V_A(\nu) &= \left(\frac{1}{6}\,\nu^2 + \frac{19}{90}\,\nu + \frac{136}{675}\right) \cdot 2^\nu + \alpha(\nu) \\ V_B(\nu) &= \left(\frac{1}{6}\,\nu^2 + \frac{7}{90}\,\nu + \frac{106}{675}\right) \cdot 2^\nu + \beta(\nu) \\ p(\nu) &= \frac{3}{5}\,2^\nu + \gamma(\nu) \\ q(\nu) &= \frac{1}{5}\,2^\nu + \gamma(\nu+1) \end{aligned}$$

where $\alpha(\nu) = (1/2)(\beta(\nu+1) + (\nu+1)\cdot\gamma(\nu))$, $\beta(\nu)$ is the periodic function given in the statement of the theorem, and $\gamma(\nu)$ is also a periodic function with period 4, given by the table below:

$\nu \bmod 4$	$\gamma(\nu)$
0	$-3/5$
1	$-1/5$
2	$3/5$
3	$1/5$

The correctness of these solutions can be readily verified by substitution.

Because of the convexity of C_T, V_B is its minimum value, completing the proof. ■

This cost compares to the cost $\left(\frac{1}{4}\,\nu^2 + \frac{1}{4}\,\nu\right)\cdot 2^\nu$ achieved by the "mid-point" construction of Figure 19.

7 Conclusion

A variety of tree layout optimization problems have been considered. Our results and those of Becker [1] show that optimization under the constraint of planarity is probably intractable. For the L_1 metric, we present efficient layout algorithms, which are even linear-time for some useful subclasses of problems. We also show closed-form solutions for complete binary trees under two simple weight functions.

For the L_2 (Euclidean) metric, convexity properties still permit general purpose optimizing techniques. Exact solutions appear to involve geometric complications and we leave open the problem of finding exact algorithms.

Another familiar metric, L_∞, is isomorphic to L_1 in up to two dimensions, so our results carry over there. In higher dimensions however the norms are essentially different, and this case we also leave open.

8 Acknowledgement

The authors are grateful to Nancy Lynch and Kurt Mehlhorn for many valuable suggestions. They also wish to thank Nancy Griffeth for several helpful discussions.

References

[1] B. Becker. On the crossing-free, rectangular embedding of weighted graphs in the plane. In *Theoretical Computer Science: 6th GI-Conference, Dortmund*, volume 145 of *Lecture Notes in Computer Science*, pages 61–72. Springer-Verlag, 1983.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R.L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[3] R. P. Brent and H. T. Kung. On the area of binary tree layouts. *Information Processing Letters*, 11:46–48, 1980.

[4] M. Brown and R. E. Tarjan. Design and analysis of a data structure for representing sorted lists. *SIAM Journal on Computing*, 9:594–614, 1980.

[5] M. J. Fischer and M. S. Paterson. Optimal tree layout. In *Proc. 12th ACM Symposium on Theory of Computing (STOC)*, pages 177–189, 1980.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractibiliy*, page 259. W. H. Freeman & Co., San Francisco, 1979.

[7] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11:329–343, 1982.

[8] C. A. Mead and L. A. Conway. *Introduction to VLSI Systems*. Addison-Wesley, Reading, Massachusetts, 1980.