Journal of Graph Algorithms and Applications

http://www.cs.brown.edu/publications/jgaa/

vol. 1, no. 2, pp. 1–15 (1997)

Optimal Algorithms to Embed Trees in a Point Set

Prosenjit Bose

School of Computer Science
Carleton University
Ottawa, Ontario, K1S 5B6
http://www.scs.carleton.ca
jit@scs.carleton.ca

Michael McAllister *Jack Snoeyink*

Department of Computer Science
University of British Columbia
Vancouver, BC, V6T 1Z4
http://www.cs.ubc.ca
mcallist@cs.ubc.ca snoeyink@cs.ubc.ca

Abstract

We present optimal $\Theta(n \log n)$ time algorithms to solve two tree embedding problems whose solution previously took quadratic time or more: rooted-tree embeddings and degree-constrained embeddings. In the rooted-tree embedding problem we are given a rooted tree T with n nodes and a set of n points P with one designated point p and are asked to find a straight-line embedding of T into P with the root at point p. In the degree-constrained embedding problem we are given a set of n points P where each point is assigned a positive degree and the degrees sum to $2n-2$ and are asked to embed a tree in P using straight lines that respects the degrees assigned to each point of P. In both problems, the points of P must be in general position and the embeddings must not have crossing edges.

Communicated by Peter Eades: submitted March, 1996; revised August, 1996.

1 Introduction

The problem of deciding whether a set of points admits a certain combinatorial structure, as well as computing an embedding of that structure on the point set, has been a recurrent theme in many fields. The list of problems falling into this category is virtually endless. We mention a few of the structures that are current topics of research.

The triangulation of a point set is a structure that has spurred much research because of its many applications in areas such as finite element methods, graphics, medical imaging, Geographic Information Systems (GIS), statistics, scattered data interpolation, and pattern recognition, to name a few [17, 18].

The combinatorial structure of interest in this paper is the tree, which is well-studied in the literature. For example, the study of spanning trees of a set of points has a long history. From a graph drawing perspective (see [6] for a survey of graph drawing), the traditional questions ask whether a (rooted or free) tree $T = (V, E)$ can be embedded in the plane such that some criterion is satisfied: e.g., that the area of the resulting embedding is small [5, 11], that the symmetry present in the tree is revealed in the embedding [13], or that T is isomorphic to the minimum-weight spanning tree [7, 15] or proximity graph [2, 3] of the points in which the vertices are embedded. In essence, the tree is given as input and one needs to construct a set of points in which to embed the tree such that it satisfies the criterion.

The two tree embedding problems that we study have a slightly different perspective: the points are given as input, and the tree may or may not be. We say that an n-node tree $T = (V, E)$ can be *straight-line embedded* onto a set of n points P, if there exists a one-to-one mapping $\phi\colon V \to P$ from the nodes of T to the points of P such that edges of T intersect only at nodes. That is, edges $(\phi(u_1), \phi(v_1)) \cap (\phi(u_2), \phi(v_2)) = \emptyset$, for all $u_1v_1 \neq u_2v_2 \in E$. We show, in the final section, that to obtain a straight-line embedding of any tree in a set of n points requires $\Omega(n \log n)$ time.

The first problem, called the rooted-tree embedding problem, was originally posed by Perles at the 1990 DIMACS workshop on arrangements: Given n points P in general position and an n-node tree T rooted at node ν, can T be straight-line embedded in P with ν at a specified point $p \in P$? Perles showed that this was possible if p was on the *convex hull of* P, which is the smallest convex set containing the points P. Pach and Törőcsik [20] showed that such an embedding is possible if p was not the *deepest point* of P, obtained by repeatedly discarding points on the convex hull. Finally, Ikebe et al. [10] showed that there was always such an embedding. All three algorithms use quadratic time. We show that one can use a deletion-only convex hull structure [4, 9] to obtain

Prosenjit Bose was partially supported by an NSERC and a Killam Postdoctoral Fellowship. Michael McAllister was partially supported by an NSERC Postgraduate Fellowship. Jack Snoeyink was partially supported by an NSERC Research Grant and a B.C. Advanced Systems Institute Fellowship.

$O(n \log^2 n)$ time and then improve this to $\Theta(n \log n)$ time. If p is the point with greatest y-coordinate then the $O(n \log^2 n)$ algorithm can embed the tree such that all paths from the root to a leaf are vertically monotone.

The second problem, degree-constrained embedding or dc-embedding, is similar, although the tree T is not specified. Consider a point set $P = \{p_1, p_2, \ldots, p_n\}$ in general position in the plane where each p_i is assigned a positive integral value d_i as its *degree*; the degrees satisfy $\sum_{i=1}^{n} d_i = 2n - 2$. Can some tree T be straight-line embedded on the set of points P such that a tree node of degree d_i maps to point p_i, for all i? Tamura and Tamura (now Ikebe) [24] showed that such a tree always exists and presented an $O(n^2 \log n)$ time algorithm to compute one. We present an optimal $\Theta(n \log n)$ time algorithm for this problem.

Similar embedding problems can be posed for planar graph embeddings in a set of points; the graphs with embeddings into a fixed point set have been characterised by Gritzmann et al. [8]. They showed that the class of outer-planar graphs is the largest class of graphs that admits an embedding in any point set and provided an embedding algorithm that runs in $O(n^2)$ time (Castañeda and Urrutia [16] later rediscovered this result). Recently, Bose [1] presented an $O(n \log^3 n)$ time embedding algorithm for this problem. Finding an optimal $O(n \log n)$ time embedding algorithm for embedding these graphs remains an open problem.

2 Hull Trees

Our algorithms for embedding trees use segments from the convex hull to avoid intersections between embedded edges. Consequently, we need efficient access to the convex hull of points. Moreover, we need the ability to delete points from the convex hull as we embed tree vertices at them. Overmars and van Leeuwen's [19] dynamic convex hull structure permits arbitrary insertion into and deletion from the set of points. We opt for hull trees [4, 9], which provide better amortised time complexities for point deletions. This section provides a brief introduction to hull trees.

A hull tree of a set of n points P stores the upper or lower convex hull of P; the entire convex hull of P can be represented by two hull trees. A hull tree for P's upper hull is a binary tree in which each leaf is a point of P and internal nodes represent an edge in the upper hull of the node's leaves (figure 1). Each internal node in the tree stores

- the upper hull edge between the convex hull of the point set at leaves in the node's left subtree and the node's right subtree.
- the number of leaves in its subtree.
- in the dc-embedding problem, each leaf has a degree value given in the problem's input. An internal node to the hull tree stores the sum of the degrees of the leaves in its subtree.

We use two of the hull tree operations described by Hershberger and Suri [9]: point deletion and set partition. The point deletion operation removes a point from the hull tree. The hull edges at internal nodes on a path from the point's leaf node to the root of the hull tree may need to be recomputed as a result of the deletion; a bottom-up merge of hulls along the path to the root accomplishes this task.

In the set partition operation, we are given a vertical line and we want to split, or partition, the hull tree into two parts: one hull tree for the points left of the vertical line and one hull tree for the remaining points. Assume that the vertical line goes through a point q. The path from the root to the leaf for q in the hull tree contains all the hull edges that cross the vertical line. Split the hull tree along this path, duplicating the path in each part to maintain connectivity. As with the deletion operation, recompute the hull edges that appear along the path in each part to get two hull trees. Finally, use the point deletion operation to remove the duplicated point from one of the hull trees as necessary.

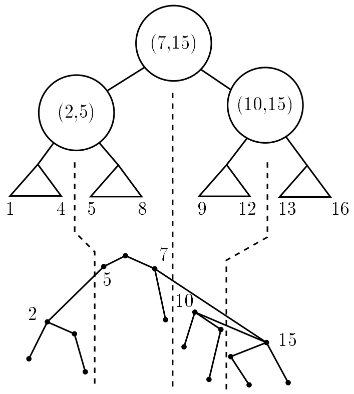

Figure 1: Top level of a hull tree (above) and upper hull edges (below).

The height of the hull tree does not increase with point deletions or set partitions.

Each of the hull tree operations uses $O(\log n)$ amortised time over any schedule of $O(n)$ operations. The set partition operation takes $O(\log n)$ time to divide the hull tree into two parts and to duplicate the path. The remaining time in the set operation is the same as in point deletion; it is the time required to recompute the hull edges along one path. Create a potential function for the hull tree by assigning each internal node a value equal to the number of hull edges that appear above the node's edge. The initial potential of the tree is $O(n \log n)$. When recomputing the hull edges, we either keep the same edge or the replacement hull edge has fewer hull edges below it, thus lowering the overall cost of the tree. If we find the replacement hull edge at a node v by walking along the two hulls of the left and right subtrees of v then updating the edges along the path takes $O(\log n + k)$ time where k is the number of hull edges over which we walked and the amount by which the potential function decreases. Consequently, the path update takes $O(\log n)$ amortised time.

3 Embedding a rooted tree with the root on the convex hull

In this section we give an algorithm for embedding a rooted tree with the root at a specified point p that is on the convex hull—general specification of p is discussed in section 4. In this restricted case, we embed the tree and preserve the order of the children about each tree node. Ikebe et al. [10] and Pach and Törőcsik [20] each provide a quadratic time algorithm whose time can be reduced to $O(n \log^2 n)$. After briefly sketching this reduction, we present an $O(n \log n)$ algorithm.

Ikebe et al. embed a tree T into a set of points P with the root at a point p on the convex hull of P by locating rays $\ell_0, \ell_1, \ldots, \ell_m$ from p such that there are exactly $|T_i|$ points between ℓ_{i-1} and ℓ_i where $T_1, \ldots, T_m$ are the subtrees of the root of T. The lines $\ell_1, \ldots, \ell_m$ are found by linear time median search. The subtrees $T_1, \ldots, T_m$ are then recursively embedded in the points between adjacent ℓ_i. This leads to an algorithm with a $\Theta(n^2)$ worst-case time.

If the points of P are placed in a convex hull maintenance structure that supports deletions in $O(\log n)$ amortised time [4, 9] then we can find the lines ℓ_i without resorting to a full median search. Let T_L be the leftmost subtree of T and let T_R be $T \setminus T_L$. Assume that $|T_L| \leq |T_R|$; reverse the roles of T_L and T_R if this is not the case. Delete, one at a time, $|T_L|$ points from the convex hull that appear as the left neighbour of p after reconstructing the convex hull every time. These are precisely the points between ℓ_0 and ℓ_1. Rebuild the convex hull maintenance structure for the deleted points; recursively embed T_L in the new convex hull structure and T_R in the convex hull structure left after the deletions. The revised complexity of the algorithm is $O(n \log^2 n)$ from the recurrence $T(n) = T(n-k)+T(k)+O(\min(k, n-k) \log n)$ where $1 \leq k \leq n-1$.

This $O(n \log^2 n)$ embedding algorithm can guarantee that each path in the embedding from the root ν to each leaf is vertically monotone if p is the highest point of P. When the algorithm recursively embeds T_L, it selects the deleted point with greatest y-coordinate as the root for T_L. Similarly, when the root ν has unit degree, the algorithm chooses the point of P with second greatest y-coordinate as the root for the single subtree of ν. The deleted points with greatest y-coordinate are found in $O(|T_L|)$ time; the point of P with second greatest y-coordinate is found in $O(\log n)$ time by keeping the points of P sorted by y-coordinate in a balanced tree and updating the tree along with the convex hull maintenance structure. The time recurrence and time complexity for the algorithm remain unchanged.

Recomputing convex hulls costs us an extra log factor in the above algorithm. To avoid this, we propose an algorithm that uses the same notion of isolating the points for one subtree but that uses only vertical separation lines and the upper hull of the points, assuming that p lies on the upper hull. When the root of the tree is not in, or immediately adjacent to, the isolated set of points, the algorithm embeds the tree along a path on the upper hull to reach the subset. By handling the leftmost or rightmost subtree of the root and then deleting the points used along the upper hull, the algorithm prevents embedded tree edges from crossing. Since all division lines are vertical, the set partition operation on hull trees in section 2 divides the hull tree in $O(\log n)$ amortised time and provides a better time complexity.

In our algorithm, each point has a label whose value is one of *left*, *right*, or *any*. When the root of the tree is a leaf, the label of the root's point indicates the direction along the upper hull where the only neighbour of the root should be embedded. For a root of higher degree, the algorithm uses recursion to

Embed a tree T, rooted at node ν, into a set of points P with distinguished point p on the upper hull of P.
Assume: P is stored in a deletion-only, upper-hull data structure $UH(P)$. The initial label for each point is *any*.

Procedure `EmbedinUH`(T, ν, P, p)
    Let ν' be the leftmost child of ν,
    Let T_L be the subtree rooted at ν', and
    Let T_R be $T \setminus T_L$.
1. If node ν has degree zero, then
        Embed node ν at point p and end.
2. If node ν has unit degree, then
        Embed node ν at point p.
        Let r be the neighbour of p on $UH(P)$ specified by p's label,
            where *any* allows either neighbour.
        Delete p from $UH(P)$.
        `EmbedinUH`$(T_L, \nu', P \setminus \{p\}, r)$ and end.
3. Else ν has degree at least two.
        Let q be the left neighbour of p on $UH(P)$, if it exists.
        Binary search $UH(P)$ for a vertical line ℓ
            with $|T_R| - 1$ points of $P \setminus \{p\}$ to its right.
    3a. If line ℓ intersects edge (p, q) as in figure 2
            Partition $UH(P)$ along ℓ into two upper hulls,
                $UH(P_L)$ to the left and $UH(P_R)$ to the right of ℓ.
            `EmbedinUH`(T_L, ν', P_L, q).
            `EmbedinUH`(T_R, ν, P_R, p).
    3b. Else if line ℓ is to the left of point q
            Label point q with *right*.
            Change the root of T to ν'.
            `EmbedinUH`(T, ν', P, q).
    3c. Else line ℓ is to the right of point p
            Restart at 3 and reverse left and right. That is, embed the rightmost subtree of ν, reverse the roles of left and right in the sidedness tests, and use the label *left* instead of *right* in case 3b,

Algorithm 1: Embedding tree T with point p on the upper hull of P

detach and embed leftmost and rightmost subtrees of the root and the labels retain path information that leads back to the embedded root of the original tree. All points are initially assigned the label *any* before invoking the recursive algorithm.

Our algorithm appears as procedure `EmbedinUH()` in Algorithm 1.

Theorem 1 *Suppose that we are given an n-node tree T with root node ν and a set of n points P in general position with the point p on the upper hull $UH(P)$.* `EmbedinUH()` *straight-line embeds T in P with ν at point p in $O(n \log n)$ time.*

Proof: We prove the algorithm correct by induction on the number of nodes in T.

If T has a single node then case 1 embeds ν at p as required.

Suppose that T has n nodes and that the algorithm correctly embeds trees of $n-1$ nodes or less for $n \geq 2$.

If ν has unit degree then T_R is only ν. Case 2 embeds ν at p as required and T_L is embedded by the induction hypothesis. The edge from ν to T_L is an edge of $UH(P)$ so it cannot intersect any edge in the embedding of T_L.

If ν has degree two or more then there are three possibilities corresponding to cases 3a, 3b, and 3c:

- line ℓ intersects edge (p, q) as shown in figure 2
- line ℓ is to the left of point q
- line ℓ is to the right of point p

If ℓ intersects edge (p, q) then step 3a embeds T_L and T_R by the induction hypothesis and the embeddings are joined by upper hull edge (p, q) that lies outside the embeddings of T_L and T_R; no intersections occur.

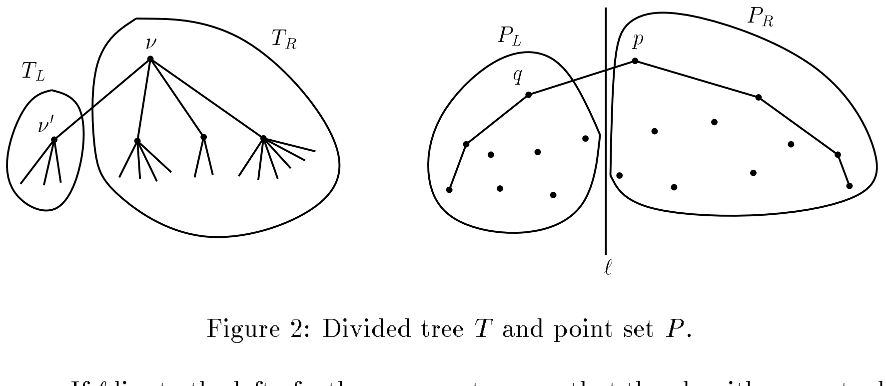

Figure 2: Divided tree T and point set P.

If ℓ lies to the left of q then we must ensure that the algorithm eventually embeds ν at p. In the recursive call of case 3b, T_R is rooted as the rightmost child of ν' in T and the new dividing line ℓ' lies to the left of ℓ. Consequently, either $T_L \setminus \nu'$ is embedded completely to the left of the vertical line through q with ν' still to be embedded at q, or the root of T is shifted left again during the embedding of T_L. Assume the former case, since the latter case eventually leads to it as the root of T continues to shift left. After the embedding of $T_L \setminus \nu'$, the remaining upper hull to the right of q is identical to $UH(P)$ right of q since all deletions occur to the left of q. In particular, p is still the right neighbour of q on the upper hull. Once $T_L \setminus \nu'$ is embedded, ν' is a leaf connected to the root of T_R. Case 2 then embeds ν' at q and the label at q, still *right* since the root shifts in T were leftward only, generates a recursive call to embed T_R into the remaining points with ν going to point p.

In the final case, ℓ lies to the right of p. If R is the rightmost subtree of ν in T then the vertical line that has exactly $|R|$ nodes to its right lies to the right of ℓ. Embedding T rooted at ν into P at point p by descending R rather than T_L will not enter case 3c again while embedding R. Instead, one of cases 3a or 3b will perform the required embedding.

For the complexity, the initial hull tree for $UH(P)$ is computed in $O(n \log n)$ time and the points of P receive their initial labels in $O(n)$ time.

Each of the algorithm steps is accomplished in $O(\log n)$ amortised time. When the root is a leaf of the tree (case 2), a point gets deleted from the hull tree in amortised $O(\log n)$ time and the height of the hull tree does not increase. Each of cases 3a and 3b fix the location of an unassigned tree node ν' at some point; this can only happen n times. Case 3a takes $O(\log n)$ time to split the hull tree while case 3b takes constant time. The direction switch of case 3c occurs at most once per subtree of a node—$O(n)$ times. The $O(\log n)$ time search for the vertical line ℓ at the start of case 3 is always paired with one of cases 3a, 3b, or 3c and is therefore done $O(n)$ times. Consequently, `EmbedinUH()` runs in $O(n \log n)$ time. ∎

A sample embedding, computed by an Ipe macro [23], appears in figure 3.

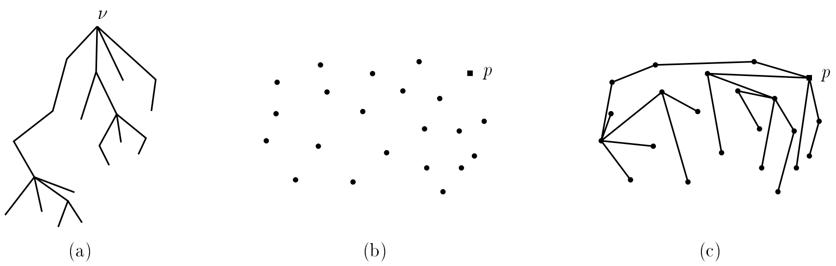

Figure 3: A rooted tree (a), a point set with distinguished point p (b), and the embedding of the tree (c).

4 Embedding a rooted tree

In this section we simplify the case analysis of Ikebe et al. [10] and apply our new algorithm to compute a straight-line embedding of T in P with the root of T at a specified point. Following Ikebe et al., we no longer attempt to preserve the ordering of children at a vertex in this embedding. We begin by improving a quadratic-time algorithm used by previous researchers [10, 20] to embed trees with two nodes mapped to adjacent hull vertices.

Theorem 2 *Suppose that we are given an n-node tree T with distinguished nodes ν and η ($\nu \neq \eta$), and a set of n points P in general position having edge (p, q) on the convex hull $CH(P)$. There is an algorithm that, in $O(n \log n)$ time, embeds T in P with ν at p and η at q.*

Proof: Assume that we have a convex hull maintenance structure for P that supports deletions of hull vertices in amortised $O(\log n)$ time; such a structure can be built initially in $O(n \log n)$ time [4, 9]. Let T_η be the induced subtree of $T \setminus \nu$ that contains η and let T_ν be the complement, $T \setminus T_\eta$.

We can find a line through p with $|T_\nu| - 1$ points on one side and q on the other side by repeatedly deleting the point of the hull adjacent to p that is different from q. When done, delete p and apply theorem 1 to embed T_ν in the deleted points with ν at p. This takes $O(|T_\nu| \log n)$ time.

Point q is on the hull of the points that remain. Let p' be the hull vertex adjacent to q where the open segment $\overline{pp'}$ does not intersect the hull. Let ν', the child of ν in subtree T_η, be the root of T_η. Recursively embed T_η with ν' at p' and η at q. The total time required for data structure building, point deletion, and tree embedding is $O(n \log n)$. ∎

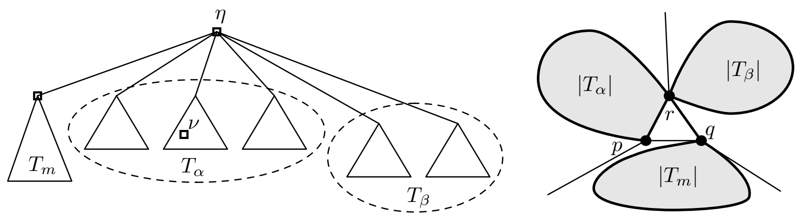

Figure 4: Partitioning T at a centroid η and embedding in P

Now we can embed a rooted tree T in a point set P with the root at a chosen point p. The basic idea is illustrated in figure 4: Use a centroid node η to partition T into a subtree T_m and two forests T_α and T_β such that we can find in P the vertices of an empty triangle $\triangle pqr$ with rays from p, q, and r that divide P into convex sets in which T_m, T_α and T_β can be embedded according to theorems 1 or 2. The partition of the tree influences the partition of the point set, and vice versa. Some special cases (when the point p is on the convex hull, the root is a centroid, or a forest is empty) are handled along the way.

Theorem 3 *Given an n-node tree T with root node ν and a set of n points P in general position, we can embed T in P with ν at a chosen point $p \in P$ in $O(n \log n)$ time.*

Proof: If p is on the convex hull $CH(P)$, then we can use the algorithm of theorem 1. Otherwise, sort the points of P radially around p.

Let η be a centroid node of T—that is, if we remove η and its incident edges from T, then we are left with connected subtrees with at most $|T|/2$ nodes [12]. For our purposes, the size of a subtree is the number of nodes other than ν that it contains. Let subtree T_m be a maximum-size subtree of $T \setminus \eta$. We now determine forests T_α and T_β and three rays from p that form angles $\leq \pi$ whose interiors contain $|T_\alpha|$, $|T_m| - 1$, and $|T_\beta|$ points, as in figure 5.

First, find a line ℓ_p through p that bisects the points—each open halfspace contains $(n-1)/2$ points. To see that such a line exists, consider the integer function $D(\theta)$ whose value is the difference between the number of points in the left and right halfspaces of a line through p at angle θ. By our general position assumption, the value of D changes by ± 1 when the line hits or passes a point. Since $D(\theta) = -D(\theta + \pi)$, the function $D(\theta)$ has a zero.

Second, choose a point $a \notin P$ on ℓ_p and a point $b \in P$ left of $\overrightarrow{pa}$ so that the interior of angle $\angle apb$ is as large as possible and contains $|T_m| - 1$ points. Recall that $|T_m|$ does not count ν if $\nu \in T_m$. There are essentially two choices—a can be chosen on either side of p, and then b is determined as the $|T_m|$th point counterclockwise around p from $\overrightarrow{pa}$. If there is a point of P on $\overrightarrow{pa}$ then perturb a into $\angle apb$. When done, the lines $\overleftrightarrow{pa}$ and $\overleftrightarrow{pb}$ determine two opposite angles as in figure 5: angle $\angle apb$ has $|T_m| - 1$ points not including b, and the opposite has at least $|T_m|$ points.

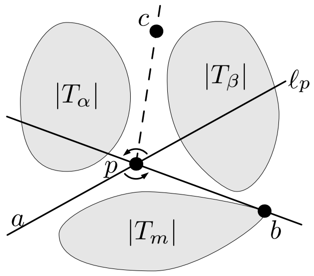

Figure 5: Find bisector ℓ_p, then $\overrightarrow{pb}$ and $\overrightarrow{pc}$

Third, enumerate the sizes of subtrees of $T \setminus \eta$ as $|T_m|$, n_1, n_2, $\ldots$, n_k, and let $N(i) = 1 + n_1 + n_2 + \cdots + n_i$. Choose a point c in the angle opposite $\angle apb$ that is the $N(i)$th point clockwise from $\overrightarrow{pa}$, for some $0 \leq i \leq k$. Such an index i exists because the angle contains at least $|T_m|$ points and $N(j+1) - N(j) = n_j \leq |T_m|$. Now let T_α consist of the subtrees of $T \setminus \eta$ with sizes $n_1, \ldots, n_i$ and let T_β be the rest of the subtrees.

In two special cases we can finish the embedding easily: If $\eta = \nu$, we embed ν into p and embed T_m, T_α, and T_β by the algorithm of theorem 1. If η has degree 2, then T_β is empty—in this case, we embed η into c and embed T_α and T_m into their appropriate angles with (c, b) connecting η to T_m. Whether ν goes with T_α or with T_m, it can be embedded at p according to theorem 2.

Otherwise, if $\eta \neq \nu$ then determine the convex hull of points inside $\angle bpc$, including b and c but not p. We can assume, without loss of generality, that node ν is in T_m or T_α. Let (q, r) be the hull edge that intersects $\overrightarrow{ap}$ and is closer to p. Note that triangle $\triangle pqr$ is empty of points of P, as in figure 6.

Finally, determine $q' \in P$ such that the open region bounded by $\overrightarrow{pa}$, $\overline{pq}$, and $\overrightarrow{qq'}$ contains $|T_m| - 1$ points; this can be done by sorting points right of $\overrightarrow{ap}$ radially around q. The slope of $\overrightarrow{qq'}$ lies between the slopes of $\overrightarrow{rq}$ and $\overrightarrow{pb}$ for the following reasons: If q' is not left of $\overrightarrow{rq}$ (dotted in figure 6) then the open region bounded by $\overrightarrow{pa}$, $\overline{pq}$, and $\overrightarrow{qq'}$ does not include any point of P from $\angle cpb$. If $\overrightarrow{qq'}$ does not intersect $\overrightarrow{pb}$, then the open region bounded by $\overrightarrow{pa}$, $\overline{pq}$, and $\overrightarrow{qq'}$ includes all points inside $\angle apb$.

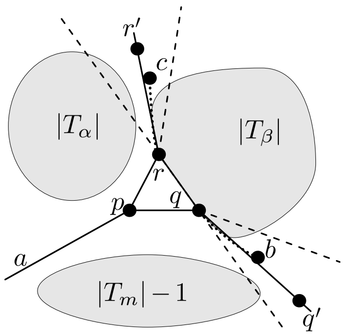

Figure 6: Finding (q, r)

Similarly, determine $r' \in P$ such that the interior of the open region bounded by $\overrightarrow{pa}$, $\overline{pr}$, and $\overrightarrow{rr'}$ contains $|T_\alpha|$ points. The slope of $\overrightarrow{rr'}$ lies between the slopes of $\overrightarrow{pc}$ and $\overrightarrow{qr}$. Thus, the three unbounded regions defined by $\triangle pqr$, $\overrightarrow{pa}$, $\overrightarrow{qq'}$, and $\overrightarrow{rr'}$ are convex.

We use theorems 1 or 2 to embed T_α, T_m, and T_β in the appropriate regions with ν at p, the root of T_m at q, and η at r. Sorting, computing convex hull structures, and embedding each take $O(n \log n)$ steps. ∎

5 Finding degree-constrained embeddings

A problem similar to the rooted-tree embedding of section 3 is to find a tree with non-crossing straight line edges in a set of points when the vertex degree for each point in the plane is given but the tree itself is not provided. Tamura and Tamura [24] called this a degree-constrained embedding (dc-embedding), observed that such an embedding exists if and only if the sum of the degrees for n points is $2n - 2$, and provided an algorithm to find a dc-embedding in $O(n^2 \log n)$ time. Using hull trees and partitions, we obtain an $O(n \log n)$ time algorithm for the same problem:

Theorem 4 *If we are given n points $P = \{p_1, p_2, \ldots, p_n\}$ in general position where each point p_i is labelled with a positive integer d_i and $\sum_{i=1}^{n} d_i = 2n - 2$ then there is an algorithm that takes $O(n \log n)$ time to find a dc-embedding on P.*

Proof: Create a deletion-only upper hull maintenance structure for the points of P as described in section 2. For the convenience of the proof, assume that the names of the points in P are sorted by x-coordinate: $p_i < p_j$ for $i < j$. Assume that $n > 1$. Finally, let $S(j) = 2j - 1 - \sum_{i=1}^{j} d_i$.

The points on the upper hull of P fall into one of three categories:

1. there is a point of unit degree and a point of degree at least two on the hull
2. all points on the hull have unit degree
3. all points on the hull have degree at least two

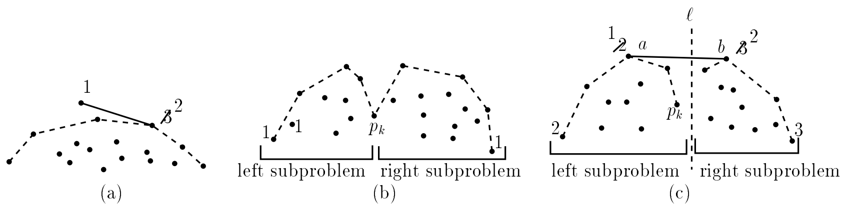

Figure 7: Three cases of recursion for the dc-embedding algorithm.

In the first case, there must exist two such points that are adjacent along the hull. Join these points by an edge, delete the point of unit degree from the hull, and decrease the degree of the other point by 1 (see figure 7a). The rest of the dc-embedding is then built recursively.

If all the hull points have unit degree then either $n = 2$, or $d_2 \geq 2$, or there exists an index k such that $S(k-1) > 0$ and $S(k) \leq 0$. If $n = 2$ then join the two vertices and stop. If $d_2 \geq 2$ then join p_1 to p_2 by a tree edge, delete p_1 from the upper hull $UH(P)$, decrease d_2 by 1, and recurse. If $d_2 = 1$ then $S(2) = 1 > 0$ and $S(n-1) = 0$ so the third condition holds for some k with $1 < k < n$. By definition, $S(k) = S(k-1) + 2 - d_k$ which implies that $d_k \geq 3$. Partition P and $UH(P)$ at p_k with p_k belonging to both smaller sets (see figure 7b). In the left subset, assign p_k a degree of $S(k-1)+1$ which is greater than zero and at most $d_k - 1$. In the right subset, assign p_k the remaining degree from d_k. Compute a dc-embedding for each subset recursively; the resulting trees will be connected through p_k.

Finally, if all the hull points have degree at least two then there exists an index k such that $S(k) = 0$ since $S(1) < 0$ and $S(n-1) \geq 0$ and the difference $S(j) - S(j-1)$ increases by at most 1 whenever p_j has unit degree. Let ℓ be a vertical line between p_k and p_{k+1} and let a and b be the left and right endpoints of the upper hull edge of P that intersects ℓ (figure 7c). Partition P and its upper hull along ℓ, join points a and b by a tree edge, decrease the degrees of points a and b by 1 each, and recursively find the dc-embedding for the subsets of P left and right of ℓ.

The time complexity of each step is $O(\log n)$ amortised time. The index k that satisfies the given conditions is found with a zero-finding search in the sequence $S(1)$, $S(2)$, ..., $S(n)$ that is stored in the hull tree. A binary search in the hull tree finds the index k in $O(\log n)$ time. The deletions and hull partitions of the steps are each done in $O(\log n)$ amortised time.

Each of the steps occurs $O(n)$ times since it either embeds a tree edge or partitions the convex hull where the partition vertex becomes a leftmost or rightmost hull vertex and is ineligible for a later partition. ■

A sample dc-embedding appears in figure 8.

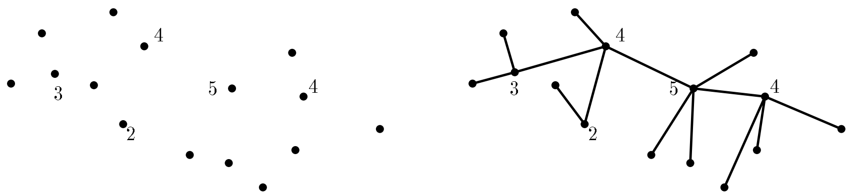

Figure 8: A sample dc-embedding. Unlabelled vertices have unit degree.

6 Lower Bounds

In this section, we provide $\Omega(n \log n)$ lower bounds on quadratic algebraic decision trees for computing a straight-line embedding of a tree onto a set of n points and for computing a dc-embedding on n points. Theorem 5 establishes the optimality of our algorithms for solving these problems. The same bound can be established with a reduction to the unit cost RAM model used by Paul and Simon [21] for their lower bound on sorting.

Theorem 5 *Finding a straight-line embedding of any tree T with n nodes into a set P of n points requires $\Omega(n \log n)$ time.*

Proof: An Euler tour of a tree T embedded into a set of n points gives a chain on $2n$ points in which no segments cross. Using such a chain from an embedding of any tree T into the point set P, a careful implementation of Melkman's algorithm [14] will then compute the convex hull of P in $O(n)$ time. The $\Omega(n \log n)$ lower bound for computing the convex hull of P [22] implies the same lower bound for finding any straight-line embedding of T into P. ■

Acknowledgments

We thank the referees for their suggestions on improving the presentation and organisation of this paper.

References

[1] P. Bose. On embedding an outer-planar graph in a point set. In *Proceedings of Graph Drawing 97*, page to appear, Rome, Italy, 1997.

[2] P. Bose, G. Di Battista, W. Lenhart, and G. Liotta. Proximity constraints and representable trees. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 340–351. Springer-Verlag, 1995.

[3] P. Bose, W. Lenhart, and G. Liotta. Characterizing proximity trees. *Algorithmica: Special Issue on Graph Drawing*, 16:83–110, 1996.

[4] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Info. Theory*, IT-31:509–517, 1985.

[5] P. Crescenzi and A. Piperno. Optimal-area upward drawings of AVL trees. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 307–317. Springer-Verlag, 1995.

[6] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[7] P. Eades and S. Whitesides. The realization problem for Euclidean minimum spanning trees is NP-hard. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 49–56, 1994.

[8] P. Gritzmann, B. Mohar, J. Pach, and R. Pollack. Embedding a planar triangulation with vertices at specified points (solution to problem e3341). *American Mathematical Monthly*, 98:165–166, 1991.

[9] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.

[10] Y. Ikebe, M. Perles, A. Tamura, and S. Tokunaga. The rooted tree embedding problem into points in the plane. *Discrete & Computational Geometry*, 11:51–63, 1994.

[11] G. Kant, G. Liotta, R. Tamassia, and I. Tollis. Area requirement of visibility representations of trees. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 192–197, Waterloo, Canada, 1993.

[12] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1973.

[13] J. Manning and M. J. Atallah. Fast detection and display of symmetry in trees. *Congressus Numerantium*, 64:159–169, 1988.

[14] A. Melkman. On-line construction of the convex hull of a simple polyline. *Information Processing Letters*, 25:11–12, 1987.

[15] C. Monma and S. Suri. Transitions in geometric minimum spanning trees. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 239–249, 1991.

[16] N. C. neda and J. Urrutia. Straight line embeddings of planar graphs on point sets. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 312–318, 1996.

[17] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, 1992.

[18] J. O'Rourke. *Computational Geometry in C.* Cambridge Univ. Press, 1994.

[19] M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comp. Sys. Sci.*, 23:166–204, 1981.

[20] J. Pach and J. Törőcsik. Layout of rooted trees. In W. T. Trotter, editor, *Planar Graphs*, volume 9 of *DIMACS Series*, pages 131–137. American Mathematical Society, 1993.

[21] W. Paul and J. Simon. Decision trees and random access machines. Logic and Algorithmics, Monograph 30, L'Enseignement Mathématique, 1987.

[22] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[23] O. Schwarzkopf. The extendible drawing editor Ipe. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C10–C11, 1995.

[24] A. Tamura and Y. Tamura. Degree constrained tree embedding into points in the plane. *Information Processing Letters*, 44:211–1214, 1992.