Optimum-width upward drawings of trees I:
Rooted pathwidth

Therese Biedl *

David R. Cheriton School of Computer Science, University of Waterloo, Waterloo,
Ontario N2L 1A2, Canada.

Abstract An upward drawing of a rooted tree is a drawing such that no parents are below their children. It is ordered if the edges to children appear in prescribed order around each vertex. It is well-known that any tree has an upward (unordered) drawing with width $\log(n+1)$. For ordered drawings, the best-known bounds for the width for binary trees is $O(\log n)$, while for arbitrary trees it is $O(2^{O(\sqrt{\log n})})$.

We present algorithms that compute upward drawings with instance-optimal width, i.e., the width is the minimum-possible for the input tree. In this first paper, we mostly study unordered drawings, where the algorithm is very simple and the drawings obtained are straight-line. We also give 2-approximation algorithms for the width of upward ordered drawings, and $O(\Delta)$-approximation where additionally the height is small. In particular any tree has an upward straight-line ordered drawing of area $O(\Delta n \log n)$.

1 Introduction

A rooted tree is an abstract way of describing a hierarchical structure, such as a family tree, an organizational chart, a file system, etc. A rooted tree consists of one node (the root) with 0 or more children which in turn are roots of rooted trees. This paper concerns algorithms to visualize such rooted trees, hence providing a tool for analyzing the structure more easily.

Formally, a drawing of a rooted tree is an assignment of points to its nodes and of polygonal curves to its edges (i.e., connections from a node to its children) such that there are no crossings or overlap. We require throughout the paper that nodes and bends of curves are placed with integral coordinates. For such a drawing, one common objective is to minimize the area of the bounding box. In particular, this means achieving a small height/width (defined as the number of rows/columns that intersect the bounding box).

Since rooted trees have a natural above/below structure, the drawings should reflect this. Thus, throughout this paper, we require all drawings to be upward, i.e., no child has y-coordinate greater than its parent, and edges are drawn as $y$-monotone curves. Such a drawing is called strictly-upward if edges are drawn as strictly $y$-monotone curves. In a strictly-upward drawing the height of the

* Supported by NSERC.
drawing must be at least the height of the tree, and so the focus has usually been on achieving a small width.

An ordered drawing is a drawing that respects a given order of the edges around each node, i.e., we are given an order of the children, in counter-clockwise order starting with the one that comes after the parent, and the drawing should respect this order.

The drawings as described will be called poly-line drawings, since edges may be polygonal curve. A straight-line drawing is a poly-line drawing for which all edges are straight-line segments, i.e., there are no bends.

Existing results The recent overview paper by Di Battista and Frati \[1\] gives a detailed list of results concerning area-bounds for drawings of trees. We will not review all of them here, but only list a few that are of particular relevance to the current paper.

There exists a simple algorithm (first published in \[6\] and reviewed in Lemma 2) that gives for any $n$-node tree a strictly-upward straight-line drawing with width at most $\log(n+1)$. This is optimal for some trees (e.g. the complete binary tree), but too large for others.

This simple algorithm gives unordered drawings. For ordered upward drawings, Chan \[5\] gave an algorithm that achieves width $O(2^{1/\epsilon} n^\epsilon)$ for any choice $\epsilon > 0$. Optimizing for the width gives a width of $O(2^{O(\sqrt{\log n})})$. For binary trees, this was improved to $O(\log n)$ width by Garg and Rusu \[10\].

All the existing algorithms for upward-drawings give bounds on the width that depend only on the size $n$ of the tree, not on the structure of the tree itself. For tree drawings that are not upward, Suderman \[14\] gave an algorithm where the smaller dimension (for which he uses the height) is at most $\frac{3}{2} \text{pw}(T)$. Here $\text{pw}(T)$ denotes the pathwidth of $T$, a graph parameter that is well-known to be a lower bound for the width and height of any planar graph drawing \[9\]. Suderman’s algorithm hence is a $\frac{3}{2}$-approximation algorithm for the height of trees in non-upward drawings. The motivation for this paper was to prove similar results for the width of upward drawings.

Our results In this series of papers, we give algorithms for upward drawings of trees for which the width is instance-optimal, i.e., tree $T$ is drawn with the smallest width that is possible for $T$. In particular our width is never bigger than in any of the previously existing algorithms for upward drawings, and for some trees (see also Lemma 5) the width improves by a factor of $\Theta(\log n)$.

For unordered drawings, the algorithm for optimum-width strictly-upward drawings is a variation of the simple algorithm in \[6\]. The only difference is in how we choose the “big” child to be put lowest; \[6\] uses here the size-heaviest child, while we use the child that requires the maximum width (which we can compute recursively). The key insight is to show that this matches a lower bound, which we give via a graph parameter that we call the rooted pathwidth of a tree.

The rooted pathwidth is useful for some constructions of ordered drawings as well. In particular, we can give 2-approximations for the width of ordered upward
drawings, which also implies that every rooted tree has an ordered upward polyline drawing of area \(O(n \log n)\). This closes the gap to the lower bound and answers Open Problem 20 in [1].

Table 1 summarizes the existing results and the improvements provided in this paper.

| Degree ∆ | Ord. | Str.-line | Width                     | Height | Ref. |
|----------|------|-----------|---------------------------|--------|------|
| General  |      |           | \(Ω(\log n)\) (for some trees) |        | [6]  |
| General  |      |           | \(≥ pw(T)\) (for all trees) |        | [9]  |
| General  |      |           | \(≥ rpw(T)\) (for all trees) |        | Lem. 1 |
| General  | ✔    |           | \(≤ \log(n + 1)\)         | \(≤ n\) | [6]  |
| General  | ✔    |           | \(≤ rpw(T)\) (Optimal)    | \(≤ n\) | Lem. 2 |
| General  | ✔    | ✔         | \(O(2^{\log n})\)        | \(O(n \cdot 2^{\log n})\) | [5]  |
| General  | ✔    | ✔         | \(≤ 2rpw(T)\)            | \(≤ 2n - 1\) | Thm. 2 |
| General  | ✔    | ✔         | \(≤ 2rpw(T)\)            | large  | Cor. 2 |
| General  | ✔    | ✔         | \(≤ (2Δ-1) \cdot rpw(T)\) | \(≤ n\) | Thm. 3 |
| Binary   | ✔    | ✔         | \(O(\log n)\)            | \(≤ n\) | [10] |
| Binary   | ✔    | ✔         | \(≤ 2rpw(T)\)            | \(≤ n\) | Cor. 3 |

Table 1: Bounds on upward drawings of rooted trees, both the existing ones and our improvements. All lower bounds hold for upward drawings, all upper bounds for strictly-upward drawings. We know \(rpw(T) ≤ \log(n + 1)\) (Lemma 3).

The paper is organized as follows. We first give the algorithm for optimum-width unordered upward drawings in Section 2. We then turn to ordered drawings and give (in Section 3) a 2-approximation algorithm for the width, and (in Section 4) a \((2Δ - 1)\)-approximation algorithm for the width of straight-line drawings with small height. We conclude with future work in Section 5.

2 Optimum-width unordered drawings

Let \(T\) be a rooted tree. For any node \(v\) of \(T\), we use \(T_v\) to denote the subtree rooted at \(v\).

Definition 1. The rooted pathwidth of \(T\) (denoted \(rpw(T)\)) is defined as follows:

\[
 rpw(T) = \begin{cases} 
 1 & \text{if } T \text{ is a single node} \\
 \min_{c_h} \max_c \{rpw(T_c) + \chi(c \neq c_h)\} & \text{otherwise}
\end{cases}
\]

Here the minimum is taken over all possible choices of one child \(c_h\) of the root, the maximum is taken over all possible choices of children \(c\) of the root, and \(\chi\) denotes the characteristic function, i.e., \(\chi(c \neq c_h)\) is 1 if \(c \neq c_h\) and 0 otherwise.

A child \(c_h\) where the minimum is achieved is called an \(rpw\)-heaviest child.
Obviously the rooted pathwidth of a rooted tree can be computed in linear time, using a bottom-up dynamic programming approach.

For some arguments it helps to know an equivalent definition of rooted pathwidth. Define a root-to-leaf path in $T$ to be any path in $T$ that connects the root to one of the leaves, i.e., one of the nodes that have no children. We call $T$ a rooted path if $T$ is a path from the root to a (unique) leaf. One can easily show the following (see the appendix for details):

**Observation 1** We have $\text{rpw}(T) = 1$ if $T$ is a rooted path, and $\text{rpw}(T) = \min_P \max_{T' \subseteq T-P} \{1 + \text{rpw}(T')\}$ otherwise. Here, the minimum is taken over all root-to-leaf paths $P$, and the maximum is taken over all subtrees $T'$ of $T-P$.

**Example:** Consider the tree in Figure 1(a). The numbers denote the rooted pathwidth of the subtree, computed with the formula in Definition 1. If we remove the root-to-leaf path $P$, then all subtrees of $T-P$ are singletons or rooted paths, and hence have rooted pathwidth 1. Therefore $\text{rpw}(T) \leq 2$ if we use the formula of Observation 1.

![Figure 1: (a) Example. (b) Lower bound. (c) “Standard” construction.](image)

**Lemma 1.** Let $\Gamma$ be any upward poly-line drawing of a rooted tree $T$. Then the width $W$ of $\Gamma$ is at least $\text{rpw}(T)$.

**Proof.** Since $\Gamma$ is an upward drawing, the root of $T$ has the maximal $y$-coordinate. Let $\ell$ be the leaf that has the minimal $y$-coordinate in $\Gamma$, breaking ties arbitrarily. Since $\Gamma$ is an upward drawing, no other node can have smaller $y$-coordinate than $\ell$. Let $P$ be the unique path from the root to $\ell$ in $T$.

If $T = P$, then $T$ is a rooted path and so $\text{rpw}(T) = 1 \leq W$. Else consider any rooted subtree $T'$ of $T-P$. The drawing $\Gamma'$ of $T'$ induced by $\Gamma$ must have width at most $W-1$, because path $P$ connects the topmost with the bottommost row in $\Gamma$, and hence any connected component of $\Gamma - P$ intersects at most $W-1$ columns. By induction, therefore $\text{rpw}(T') \leq W - 1$ for all subtrees $T'$ of $T-P$, and so $\text{rpw}(T) \leq W$. \[\square\]
Lemma 2. Any rooted tree $T$ has a strictly-upward drawing of width at most $\text{rpw}(T)$.

Proof. We modify the algorithm of [6], which draws any rooted tree $T$ such that the root is at the top-left corner. This is straightforward if $T$ is a single node. So assume $T$ has children $c_1, \ldots, c_k$ with subtrees $T_1, \ldots, T_k$, and draw $T_i$ recursively. Place the drawings of $T_k, T_{k-1}, \ldots, T_2, T_1$, one above the other, such that the root of $T_i$ is in column 2 for $i = k, \ldots, 2$ and in column 1 for $i = 1$. See Figure 1(c). Clearly we can connect $v$ to all its children without crossing.

If $W(T)$ denotes the width of the drawing of $T$, then
$$W(T) = \begin{cases} 1 & \text{if } T \text{ is a single node} \\ \max_i \{W(T_i) + \chi(i \neq 1)\} & \text{otherwise} \end{cases}$$

The width hence depends on the choice of child $c_1$. If $c_1$ is the size-heaviest child (i.e., the one with the most nodes in its subtree), then $n(T_i) \leq (n-1)/2$ for all $i \neq 1$, and one easily shows by induction that $W(T) \leq \log(n+1)$. (This is the approach taken by [6].) But if instead $c_1$ is the rpw-heaviest child, then $\text{rpw}(T_i) \leq \text{rpw}(T) - 1$ for all $i \neq 1$. By induction $T_i$ (for $i \neq 1$) is drawn with width $\text{rpw}(T_i) \leq \text{rpw}(T) - 1$, and $T_1$ is drawn with width $\text{rpw}(T_1) \leq \text{rpw}(T)$, therefore $W(T) \leq \text{rpw}(T)$.

Observe that the height of the drawing is $n$, since every row intersects exactly one node. Since the rooted pathwidth (and with it the rpw-heaviest child for each node) can be found in linear time, we therefore have:

Theorem 1. There exists a linear-time algorithm to create for any rooted tree $T$ a strictly-upward straight-line drawing of optimal width and height $n$.

For the purpose of comparing our algorithms to existing results, it is useful to bound the size of the rooted pathwidth.

Lemma 3. Any tree $T$ with $\text{rpw}(T) = r$ has at least $2^r - 1$ nodes and at least $2^{r-1}$ leaves. In particular, $\text{rpw}(T) \leq \log(n+1)$.

Proof. Clearly this holds if $T$ is a single node and $r = 1$, so assume the root has children. If one child $c$ has $\text{rpw}(T_c) = r$, then the claim holds by induction for $T_c$ and hence also for $T$. Otherwise, by definition of $\text{rpw}(T)$ there must be at least two children $c_1, c_2$ with $\text{rpw}(T_{c_j}) = r - 1$. Applying induction to both and combining the bounds (and adding the root) gives the result.

The rooted pathwidth is also closely related to two other graph parameters: the pathwidth and the depth of a heaviest-path decomposition. We state the results here; formal definition of these concepts and proofs can be found in the appendix.

Lemma 4. For any rooted tree $T$, we have $\text{pw}(T) \leq \text{rpw}(T) \leq 2 \text{pw}(T) + 1$, where $\text{pw}(T)$ denotes the pathwidth of the tree.

Lemma 5. For any rooted tree $T$, we have $\text{rpw}(T) \leq \text{hpd}(T)$, where $\text{hpd}(T)$ denotes the depth of the heaviest-path decomposition of $T$. There exists an infinite number of binary trees $T$ with $\text{rpw}(T) = 2$ and $\text{hpd}(T) \in \Omega(\log n)$. 

Observe that the height of the drawing is $n$, since every row intersects exactly one node. Since the rooted pathwidth (and with it the rpw-heaviest child for each node) can be found in linear time, we therefore have:

Theorem 1. There exists a linear-time algorithm to create for any rooted tree $T$ a strictly-upward straight-line drawing of optimal width and height $n$.

For the purpose of comparing our algorithms to existing results, it is useful to bound the size of the rooted pathwidth.

Lemma 3. Any tree $T$ with $\text{rpw}(T) = r$ has at least $2^r - 1$ nodes and at least $2^{r-1}$ leaves. In particular, $\text{rpw}(T) \leq \log(n+1)$.

Proof. Clearly this holds if $T$ is a single node and $r = 1$, so assume the root has children. If one child $c$ has $\text{rpw}(T_c) = r$, then the claim holds by induction for $T_c$ and hence also for $T$. Otherwise, by definition of $\text{rpw}(T)$ there must be at least two children $c_1, c_2$ with $\text{rpw}(T_{c_j}) = r - 1$. Applying induction to both and combining the bounds (and adding the root) gives the result.

The rooted pathwidth is also closely related to two other graph parameters: the pathwidth and the depth of a heaviest-path decomposition. We state the results here; formal definition of these concepts and proofs can be found in the appendix.

Lemma 4. For any rooted tree $T$, we have $\text{pw}(T) \leq \text{rpw}(T) \leq 2 \text{pw}(T) + 1$, where $\text{pw}(T)$ denotes the pathwidth of the tree.

Lemma 5. For any rooted tree $T$, we have $\text{rpw}(T) \leq \text{hpd}(T)$, where $\text{hpd}(T)$ denotes the depth of the heaviest-path decomposition of $T$. There exists an infinite number of binary trees $T$ with $\text{rpw}(T) = 2$ and $\text{hpd}(T) \in \Omega(\log n)$. 

Observe that the height of the drawing is $n$, since every row intersects exactly one node. Since the rooted pathwidth (and with it the rpw-heaviest child for each node) can be found in linear time, we therefore have:

Theorem 1. There exists a linear-time algorithm to create for any rooted tree $T$ a strictly-upward straight-line drawing of optimal width and height $n$.

For the purpose of comparing our algorithms to existing results, it is useful to bound the size of the rooted pathwidth.
3 2-approximations for ordered drawings

One naturally wonders whether we can create ordered drawings whose width is optimal? This is feasible, but quite complicated, and will be studied in a separate paper [4]. However, in this section we show that 2-approximations of the width of ordered upward drawings are easy to achieve.

Theorem 2. Any rooted tree \(T\) has an ordered strictly-upward poly-line drawing of width at most \(2\) \(\text{rpw}(T) - 1\). Furthermore, every edge is rooted \(x\)-monotonically, and the height is at most \(2n - \ell(T)\), where \(\ell(T)\) denotes the number of leaves of \(T\). It can be found in linear time.

Proof. We again impose the additional constraint that the root must be at the top-left corner of the drawing. A symmetric construction also gives such a drawing where the root is at the top-right corner of the drawing.

Clearly any single-node tree has a drawing of width 1 = \(2\) \(\text{rpw}(T) - 1\) and height 1 = \(2n - \ell(T)\) that satisfies all conditions. If \(n > 1\), then let \(v_r\) be the root and \(c_1, \ldots, c_d\) be the children. By definition of rooted pathwidth at most one of the children \(c\) satisfies \(\text{rpw}(T_c) = \text{rpw}(T)\); let \(c_h\) be this child. (If there is no such child, then set \(c_h := c_1\).) We distinguish cases by \(h\).

Case 1: \(h = 1\), i.e., \(\text{rpw}(T_{c_j}) < \text{rpw}(T)\) for all \(j > 1\): In this case, use the “standard” construction, i.e., recursively obtain drawings of each \(T_{c_j}\), \(j = 1, \ldots, d\), that satisfy all conditions and for which the root is placed at the top-left corner, and combine as in Figure 1(c). One easily verifies all conditions.

Case 2: \(h > 1\): Place the root with \(x\)-coordinate 1. Set \(W := 2\) \(\text{rpw}(T) - 1\); we aim to create a drawing within columns 1, \ldots, \(W\). By induction \(T_{c_h}\) can be drawn with width \(W\) such that \(c_h\) is at the top right corner. For all \(j \neq h\), draw \(T_{c_j}\) recursively such that \(T_{c_j}\) is at the top-left corner and notice that this drawing has width at most \(2\) \(\text{rpw}(T_{c_j}) - 1 \leq 2(\text{rpw}(T) - 1) - 1 = W - 2\).

Place \(T_{c_{d}}, \ldots, T_{c_{h-1}}\), in this order from top to bottom, in columns 2, 3, \ldots, \(W - 1\), flushed left.

Place a bend for edge \((v_r, c_h)\) in column 2, below all that was drawn thus far. Place another bend one row below and in column \(W\).

Next, place \(T_{c_{h-1}}, \ldots, T_{c_2}\), in this order from top to bottom, and flush left with column 2, using columns 2, 3, \ldots, \(W - 1\). The topmost of these drawings can share the row of the second bend, since it is in a different column.

In the penultimate step, place \(T_{c_1}\) flush left with column 1, and draw the edge to it vertically downward from the root. Finally, place \(T_{c_h}\) below all the drawings, but flush right so that \(c_h\) is in column \(W\), and connect \(c_h\) to the second bend vertically.

 ![Figure 2: 2-approximation.](image-url)
In both cases the height of the drawing is the sum of the heights of the subtrees, plus one row for the root and (possibly) one row for the first bend. Hence it is at most

\[ 1 + \sum_{i=1}^{d} (2n(T_{c_i}) - \ell(T_{c_i})) + 1 = 2(n - 1) - \ell(T) + 2 = 2n - \ell(T) \]

as desired.

Since \( rpw(T) \leq \log(n + 1) \) by Lemma 3, we hence have:

**Corollary 1.** Every rooted tree has an ordered strictly-upward poly-line drawing of area \( O(n \log n) \).

This solves Open Problem 20 posed by Di Battista and Frati [1], which asked to close the gap between the previous best upper bound of \( O(n^{1+\varepsilon}) \) [5] and the lower bound of \( \Omega(n \log n) \).

**Corollary 2.** Every rooted tree \( T \) has an ordered strictly-upward straight-line drawing of width at most \( 2\cdot rpw(T) - 1 \).

**Proof.** By the previous theorem \( T \) has a strictly-upward ordered poly-line drawing of this width such that edges are drawn \( x \)-monotonically. It is known [7,12] that such a drawing can be turned into a straight-line drawing without increasing the width. Neither of these references discusses whether strictly-upward drawings remain strictly-upward, but it is not hard to show that this can be done: essentially each subtree needs to “slide down” far enough to allow bends to be straightened out.

Notice that Corollary 2 does not bound on the height. Indeed, the transformations to straight-line drawings might increase the height exponentially in general (see [2]), and no better bounds for trees seem apparent. The area of our straight-line drawing is hence no improvement on the area of \( O(4\sqrt{\log n}) \) achieved by Chan [5], and it remains open to find straight-line ordered upward-drawings of area \( O(n \log n) \) or to give examples where they cannot exist.

## 4 Straight-line ordered drawings with linear height

In 2003, Garg and Rusu [10] showed that every binary tree has an upward straight-line drawing of width \( O(\log n) \) and height at most \( n \). However, their construction does not generalize to higher degrees (unless one drops “upward”). We now give a different construction that achieves these bounds for any tree that has bounded maximum degree.

**Theorem 3.** Every rooted tree \( T \) has a strictly-upward ordered straight-line drawing of width \( (2\Delta - 1)(rpw(T) - 1) + 1 \) and height at most \( n \), where \( \Delta \) is the maximum number of children of a node. It can be found in linear time.

In particular any rooted tree has a strictly-upward ordered straight-line drawing of area \( O(\Delta n \log n) \); this is an improvement over the area-bound of \( O(4\sqrt{\log n}) \) by Chan [5] for small (but more than constant) values of \( \Delta \).
The rest of this section gives the construction that proves this theorem. For ease of description, define shortcuts $r := \text{rpw}(T)$ and $W(i) := (2\Delta - 1)(i - 1) + 1$; we aim to create drawings of width at most $W(r)$. As before we create drawings where the root is in the top-left corner, and a symmetric construction places the root in the top-right corner.

If $r = 1$ then $W(1) = 1$ and $T$ is a rooted path that we can draw in a single column as desired. So assume $r > 1$, which means that $\Delta \geq 2$ and that the root has children $c_1, \ldots, c_d$, $1 \leq d \leq \Delta$. Let $c_h$ be the child, if any, with $\text{rpw}(T_{c_h}) = r$. If there is no such child, then set $c_h := c_1$.

Case 1: $h = 1$: In this case, draw the tree as in the “standard” construction, i.e., recursively obtain drawings of each $T_{c_j}$, $j = 1, \ldots, d$, with $c_j$ in the top-left corner and combine as in Figure 1(c). The drawing of $T_{c_1}$ has width at most $W(r)$ and the drawing of each $T_{c_j}$ for $j > 1$ has width at most $W(r - 1) \leq W(r) - 1$, to which we add at most one unit width. Clearly all conditions are satisfied.

Case 2: $h > 1$: The construction in this case is much more complicated (and quite different from Garg and Rusu’s). We use $W(r) = W(r - 1) + 2\Delta - 1$ columns for our drawing, and split them into 3 groups as follows:

- The leftmost $\Delta - 1$ columns are called left-detour columns. The rightmost of these is called the left-overhang column.
- The next $W(r - 1) + 1$ columns are the middle columns; the leftmost and rightmost of these are also called left-path and right-path column.
- The last $\Delta - 1$ columns are called the right-detour column. The leftmost of these is called the right-overhang column.

Figure 3 sketches the construction. The main tool is to use a path $P = v_0, v_1, v_2, \ldots$ which is a root-to-leaf path that minimizes $\max_{T' \subseteq T - P} \{\text{rpw}(T') + 1\}$. We know that this minimum is $r = \text{rpw}(T)$ by Observation 1. Also note that $v_1$ must be child $c_h$, and so in particular $v_1$ is not the leftmost child of $v_0$ by case assumption.

We first outline the idea. To place path $P$, we split it into many sub-paths of length at least 2. These sub-paths are alternatingly placed in the left-path column (or nearby) and the right-path column (or nearby). Whenever possible, subtrees of these paths are placed in the middle columns. However, this is not always possible for the top-most and bottom-most node of a sub-path. For these, we use the detour-columns, either for placing the node or for placing its children. However, the subtrees of these nodes or children cannot be placed here; instead we put them “much farther down”, namely, at such a time when path $P$ has veered to the other side and therefore the middle columns are accessible.

The precise placement of path $v_0, v_1, v_2, \ldots$ is as follows. Place the root $v_0$ in the top left corner, set $i = 1$ and $\ell_1 = 0$. (Generally $\ell_i$ will be the index of the bottommost vertex of the $i$th sub-path on the left, and $r_i$ will be the index of the bottommost vertex of the $i$th sub-path on the right.) Now repeat:

- $v_{\ell_{i}+1}$ is placed in the right-overhang column, one row below $v_{\ell_{i}}$. 
$v_0 = v_{\ell 1}$

Figure 3: The construction for ordered straight-line drawings if the child with the same rooted pathwidth is not the leftmost child. Path $P$ is purple (dashed).

- $v_{\ell i+2}$ is placed in the right-path column, some rows below. ("Some rows below" means that the precise row depends on the heights of some subtrees and will be determined later.)
- While $v_j$ is the rightmost child of $v_{j-1}$ (for $j = \ell_i + 3, \ell_i + 4, \ldots$), place it in the right-path column, some rows below.
- Let $r_j \geq \ell_i + 2$ be the maximal index for which $v_{r_j}$ was placed in the right-path column. So $v_{r_j+1}$ is not the rightmost child of $v_{r_j}$.
- Place $v_{r_j+1}$ in the left-overhang column, one row below $v_{r_j}$.
- Place $v_{r_j+2}$ in the left-path column, some rows below.
- While $v_j$ is the leftmost child of $v_{j-1}$ (for $j = r_i + 3, r_i + 4, \ldots$), place it in the left-path column, some rows below.
- Let $\ell_{i+1} \geq r_i + 2$ be the maximal index for which $v_{\ell_{i+1}}$ was placed in the left-path column.
- Update $i := i + 1$, and repeat until we reach the end of path $P$.

For any non-leaf node $v$ on $P$, let the left [right] children of $v$ be all those children of $v$ that are strictly left [right] of the child of $v$ on $P$. We now explain how to place all the subtrees that are attached at right children of nodes $v_{\ell_i}, \ldots, v_{\ell_{i+1}-1}$, for $i = 1, 2, \ldots$. The subtrees at left children are placed symmetrically.

1. We start at $v_{\ell_i}$. The right children of $v_{\ell_i}$ are placed, in order, in the row below $v_{\ell_i}$ and in the right-detour column. By choice of $\ell_i$ (or, for $i = 1$, by case assumption) node $v_{\ell_i+1}$ is not the leftmost child of $v_{\ell_i}$. So $v_{\ell_i}$ has at least one left child, therefore at most $\Delta - 2$ right children, which means that there are sufficiently many right-detour columns for placing the right children as well as $v_{\ell_i+1}$. We can connect these children to $v_{\ell_i}$ (drawn curved in Figure 3 for increased visibility.) The subtrees for these children are not being placed yet; this will happen in Step 6.

2. The next node is $v_{\ell_i+1}$, which is in the right-detour column one row below $v_{\ell_i}$. The subtrees at its right children will be placed in Step 6.

3. The next nodes are $v_{\ell_i+2}, \ldots, v_{r_i-1}$. By choice of $r_i$ these nodes do not have right children. The rows for these nodes (as well as $v_{r_i}$) are determined by the algorithm that places subtrees of left children (which we presume to operate in parallel); namely, when it reaches its Step 7.

4. The next node is $v_{r_i}$ placed in the right-path column. We place the subtrees of its right children with the symmetric version of the standard construction. Thus recursively obtain for each such subtree a drawing of width at most $W(i-1)$ with the child in the top-right corner. Place these, in order, in the rows below $v_{r_i}$ and in the columns to its left (except for the last child, which shares the column with $v_{r_i}$). This fits within the middle columns since there are $W(r-1) + 1$ middle columns and $v_{r_i}$ is in the rightmost of these.

5. Next comes node $v_{r_i+1}$, in the row below $v_{r_i}$ and the left-overhang column. This node might share a row with some right child of $v_{r_i}$ but uses a different column. The subtrees at $v_{r_i+1}$’s right children will be placed in Step 6.

6. Now we place all the subtrees that were deferred earlier. First place, flush right with the right-path column, the subtrees of right children of $v_{\ell_i+1}$, drawn with their roots in the top-right corner. Recall that $v_{\ell_i+1}$ was placed in the right-overhang column while its children are now in the right-path column, which is adjacent. Hence the edges can be drawn with straight-line segments (shown again with curves in Figure 3).
Next, we place the subtrees of right children of $v_{\ell i}$, parsing them in left-to-right order. If $c$ is such a child, then $c$ was placed much higher up already in one of the right-detour columns. Let $g_1, \ldots, g_d$ be the children of $c$ (hence grand-children of $v_{\ell i}$). For each $g_i$ create a drawing of $T_{g_i}$ with $g_i$ in the top-right corner. Place these drawings in the middle columns as well as the right-detour columns so that $g_1, \ldots, g_d$ are one column to the left of $c$. Then $c$ can be connected with straight lines (shown again with curves). Finally place the subtrees of right children of $v_{r i+1}$. Recursively draw each such subtree with the root in the top-left corner. Place these, in order, flush left with the left-path column, and draw the edges to $v_{r i+1}$ as straight-line segments.

7. Next come nodes $v_j$ for $j = r_i + 2, r_i + 3, \ldots, \ell_i + 1 - 1$. For each $j$, place $v_j$ in the next row (i.e., the first row below what was drawn so far) and in the left-path column. Recursively draw the subtree of each right child of $v_j$ with the root in the top-left corner. Place these, in order, flush left with the column that is one right of the left-path column.

8. Finally put $v_{\ell i+1}$ in the next row; and go to Step 1 with the next $i$.

This ends the description of the construction, which has width $W(r)$ as desired. All rows contain nodes, so the height is at most $n$. This proves Theorem 3.

We note that for binary trees, our construction gives a width of at most $3rpw(T)$, hence a 3-approximation. This can be turned into a 2-approximation by observing that $v_{r_i}$ can have at most one right child for binary trees. Hence no right child of $v_{r_i}$ needs to use the left-path column. Symmetrically no left child of $v_{\ell i}$ needs to use the right-path column. Observe that then in all rows at least one column is unused. We can hence reduce the number of middle columns to $W(r-1)$ and still fit all subtrees (some right children then use the right-overhang column). Hence the new width $W'$ satisfies $W'(1) = 1$ and $W'(r) = W'(r-1) + 2(\Delta - 1) = W'(r-1) + 2$, which one computes easily to be $W'(r) = 2rpw(T) - 1$ for binary trees.

**Corollary 3.** Any binary tree has a strictly-upward straight-line drawing of width at most $2rpw(T) - 1$ and height $O(n)$.

**5 Conclusion**

In this paper, we gave algorithms for strictly-upward drawings of trees that have instance-optimal width. The key ingredient was the definition of the rooted pathwidth, a graph parameter that is closely related to the pathwidth, but which exactly encapsulates the width of an upward drawing of a tree. We then used this parameter to find ordered strictly-upward drawings that approximate the optimal width and have linear height; the approximation factor is 2 for poly-line drawings and $2\Delta - 1$ for straight-line drawings.

As we will discuss in a forthcoming paper, one can find upward ordered poly-line drawings with optimal width; this does not use the rooted pathwidth but instead a different measure which is much harder to define, compute and analyze.

Many open problems remain:
– We minimize the width, but not the area. Minimizing the area of a straight-line drawing is NP-hard, even under many restrictions on the graph or the drawing (see \cite{11,3}), but it is not known whether minimizing the area is NP-hard for upward drawings of trees.
– Does every rooted tree have a strictly-upward straight-line ordered drawing of area $O(n \log n)$? We believe this to be true, but could show it only for trees of constant maximum degree.
– Can we achieve the optimum width for drawings that need not be upward? Note that the rooted pathwidth is not a lower bound here. The (unrooted) pathwidth is a lower bound on the width of such drawings, but this is not always tight \cite{14}. Is there another graph parameter that matches exactly the width of (not necessarily upward) drawings of trees?

References

1. G. Di Battista and F. Frati. A survey on small-area planar graph drawing, 2014. arXiv: 1410.1006.
2. T. Biedl. Height-preserving transformations of planar graph drawings. In Graph Drawing (GD'14), volume 8871 of Lecture Notes in Computer Science, pages 380–391. Springer, 2014.
3. T. Biedl. On area-optimal planar grid-drawings. In International Colloquium on Automata, Languages and Programming (ICALP ’14), volume 8572 of Lecture Notes in Computer Science, pages 198–210. Springer-Verlag, 2014.
4. T. Biedl. Optimum-width upward drawings of trees II: Ordered drawings. In preparation, 2015.
5. Timothy M. Chan. A near-linear area bound for drawing binary trees. Algorithmica, 34(1):1–13, 2002.
6. P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. Comput. Geom., 2:187–200, 1992.
7. Peter Eades, Qing-Wen Feng, and Xuemin Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In Graph Drawing (GD’96), volume 1190 of Lecture Notes in Computer Science, pages 113–128. Springer, 1997.
8. J.A. Ellis, I. Hal Sudborough, and J.S. Turner. The vertex separation and search number of a graph. Inf. Comput., 113(1):50–79, 1994.
9. S. Felsner, G. Liotta, and S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. Journal of Graph Algorithms and Applications, 7(4):335–362, 2003.
10. A. Garg and A. Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. Int. J. Comput. Geometry Appl., 13(6):487–505, 2003.
11. M. Krug and D. Wagner. Minimizing the area for planar straight-line grid drawings. In Graph Drawing (GD’07), volume 4875 of LNCS, pages 207–212. Springer-Verlag, 2007.
12. J. Pach and G. Tóth. Monotone drawings of planar graphs. Journal of Graph Theory, 46(1):39–47, 2004.
13. D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. Journal of Computer And System Sciences, 26:362–392, 1983.
14. M. Suderman. Pathwidth and layered drawings of trees. International Journal of Computational Geometry and Applications, 14(3):203–225, 2004.
A Rooted pathwidth and other parameters

In this section we relate the rooted pathwidth to other parameters of a tree.

**Root-to-leaf paths:** Let $P$ be a root-to-leaf path in $T$, i.e., a path from the root to some arbitrary leaf. Removing $P$ splits $T$ into subtrees. We now claim that if we choose $P$ suitably, then all these subtrees have smaller rooted pathwidth, and show:

**Observation 1** We have

$$rpw(T) = \begin{cases} 1 & \text{if } T \text{ is a rooted path} \\ \min_P \max_{T' \subseteq T - P} \{1 + rpw(T')\} & \text{otherwise} \end{cases}$$

**Proof.** We show ‘$\geq$’ by induction on the height of the tree. Clearly the claim holds for a single-node tree, so assume the root has children. Let $P$ be the path obtained by going from the root to its rpw-heaviest child, and from there to its rpw-heaviest child, etc., until we reach a leaf. Any subtree $T'$ of $T - P$ then corresponds to tree $T_c$ for a node $c$ which is not on $P$, but its parent $v$ is on $P$. Since $c$ was not the rpw-heaviest child of $v$, we have $rpw(T_c) < rpw(T_v) \leq rpw(T)$, hence $\max_{T' \subseteq T - P} \{1 + rpw(T')\} \leq rpw(T)$. The minimum over all choices of path can only be smaller.

For the other direction, let $P$ be the path that minimizes $r := \max_{T' \subseteq T - P} \{1 + rpw(T')\}$, and let $c_h$ be the child of the root that belongs to $P$. Then any child $c \neq c_h$ of the root gives rise to a subtree $T' = T_c$ of $T - P$, hence $1 + rpw(T_c) \leq r$. Also, $rpw(T_{c_h}) \leq r$ by induction, since $P$ (minus the root) can be used as a path for $T_{c_h}$. Therefore $\max_c \{rpw(T_c) + \chi(c \neq c_h)\} \leq r$ and the minimum over all choices of $c_h$ can only be smaller. \qed

**Pathwidth:** The *pathwidth* $pw(G)$ of a graph $G$ is a well-known graph parameter; it is the smallest integer $k$ such that $G$ is a subgraph of a $(k + 1)$-colorable interval graphs. For trees, the pathwidth can also be described via a decomposition into paths; see [8,14]. Namely

$$pw(T) = \begin{cases} 0 & \text{if } T \text{ is a single node} \\ \min_P \max_{T' \subseteq T - P} \{1 + pw(T')\} & \text{otherwise} \end{cases}$$

where the minimum is taken over all paths $P$. As in [14] we call the path $P$ where the minimum is achieved the *main path*. Note that the recursive formula is the same as in Observation 1 except that the path $P$ is not restricted to end at the root. A simple proof by induction hence shows that $pw(T) \leq rpw(T)$. At the other end, we can show:

**Lemma 4.** For any rooted tree $T$, we have $rpw(T) \leq 2pw(T) + 1$. 

Proof. This was essentially shown by Suderman [14] (he also gives credit to Dujmović and Wood) without using the term “rooted pathwidth”. In the second half of the proof of his Lemma 7, he creates tree-drawings of height at most $2\text{pw}(T)$. An inspection of the construction shows that it gives upward drawing after $90^\circ$ rotation, except at subtrees with pathwidth 1 (which could be drawn upright if we allowed one extra unit.) By Lemma 1 hence $\text{rpw}(T) \leq 2\text{pw}(T) + 1$.

For completeness’ sake, we give here an independent proof of this result, using the same idea as implicit in Suderman’s algorithm [14]. If $\text{pw}(T) = 0$, then $T$ is a single node and $\text{rpw}(T) = 1 = 2\text{pw}(T) + 1$, so the claim holds. If $\text{pw}(T) \geq 1$, then let $P$ be a main path of $T$. See also Figure 4. We may, after expanding $P$ if needed, assume that the ends of $P$ are at the root or at a leaf. Let $v$ be the node of $P$ that is closest to the root, and write $P = P_1 - v - P_2$ for two paths $P_1$ and $P_2$. By definition any subtree $T'$ of $T - P$ has $\text{pw}(T') \leq \text{pw}(T) - 1$ and therefore $\text{rpw}(T') \leq 2\text{pw}(T) - 1$.

Let $P_0$ be the path from the root to $v$. Let $P' := P_0 - v - P_1$ consists of the path from the root to $v$, followed by one part of the $\text{pw}$-main path of $T$. We use $P'$ as the path in Observation 1 and hence must study the rooted pathwidth of any subtree $T'$ of $T - P'$. If $T'$ is also a subtree of $T - P$, then as argued above $\text{rpw}(T') \leq 2\text{pw}(T) - 1$. If $T'$ is not a subtree of $T - P$, then $T'$ necessarily must contain $P_2$; call this subtree $T_2$.

One can show that $\text{rpw}(T_2) \leq 2\text{pw}(T)$ as follows. Use path $P_2$ as the path in Observation 1, we hence must study the rooted pathwidth of any subtree $T''$ of $T_2 - P_2$. But any such subtree contains no nodes of $P$ and hence is a subtree of $T - P$. By the above discussion therefore $\text{rpw}(T'') \leq 2\text{pw}(T) - 1$. Therefore $\text{rpw}(T_2) \leq \max_{T''} \{1 + \text{rpw}(T'')\} \leq 2\text{pw}(T)$.

Putting it all together, we know that $\text{rpw}(T') \leq 2\text{pw}(T)$ for all subtrees $T'$ of $T - P$, and by Observation 1 therefore $\text{rpw}(T) \leq 2\text{pw}(T) + 1$. □

![Figure 4: The main path $P_1 - v - P_2$ can be used to show $\text{rpw}(T_2) \leq 2\text{pw}(T)$ and therefore $\text{rpw}(T) \leq 2\text{pw}(T) + 1$.](image-url)
Heavy-path decompositions: The heavy-path decomposition, first introduced by Sleator and Tarjan [13], is a method of splitting a tree into paths such that any root-to-leaf path encounters $O(\log n)$ of these paths. Recall that the size-heaviest child of the root is the child whose subtree contains the most nodes (breaking ties arbitrarily). The heaviest path is obtained by going from the root to a leaf by always going to the size-heaviest child. If we remove the heaviest path and recurse in the children, then after some number of recursions the remaining tree is empty; this number of recursions is called the heaviest-path depth (and denoted $hpd(T)$). Formally,

$$hpd(T) = \begin{cases} 1 & \text{if } T \text{ is a single node} \\ \max_c \{hpd(T_c) + \chi_{c \neq c_h}\} & \text{otherwise} \end{cases}$$

where the maximum is taken over all children $c$ of the root, and $c_h$ is the size-heaviest child. Note that the recursive formula is very similar to, but more restrictive, than the one in Definition 1; by induction one easily shows that $rpw(T) \leq hpd(T)$ for all rooted trees $T$. This is far from tight for some trees.

Lemma 5. There exists an infinite number of binary trees $T$ with $rpw(T) = 2$ and $hpd(T) \in \Omega(\log n)$.

Proof. Let $T_1$ be a single node. For $i > 1$, let $T_i$ consist of a root with left subtree $T_{i-1}$ and right subtree a rooted path of length $|T_{i-1}| + 1$. Clearly $rpw(T_i) = 2$, using the left child as $c_h$ in Definition 1 since the right subtrees are rooted paths and hence have rooted pathwidth 1. But the right child is the size-heaviest child, and therefore $hpd(T_i) = 1 + hpd(T_{i-1}) = i$. Since $|T_i| = 2|T_{i-1}| + 2 = 2^{i+1} - 2$, the result follows. \qed

In particular, the above trees can be drawn with width 2, but the algorithm of [6] draws them with width $\Theta(\log n)$. 