1 Tree Drawing

The first two lectures focused primarily on methods for drawing trees. We looked at some desirable drawing properties when considering a tree drawing problem, and numerous schemes for drawing a binary tree with integer coordinates. The first interesting idea was a divide and conquer scheme that produces what is called a tidy drawing [3]. Afterwards, we discussed HV-Trees, leading to Chan et al.’s Recursive Winding algorithm [1]. We then examined a few alternative methods that apply to non-binary trees, such as Teoh and Ma’s Ring representation [4] and Lamping et al.’s Hyperbolic Trees [2].

1.1 Desirable Properties

There are many important properties of a tree drawing that should be considered when measuring its value, such as the following:

- Top-down ordering (parents drawn with higher $y$-coordinate than children)
- Minimal edge crossings
- Straight line edges
- Locally symmetric (A node’s two children are at the same $y$-coordinate and are equally offset horizontally from their parent)
- Minimal total/variation of/max edge length
- Minimal total/variation of/max edge bends (kinks or elbows in the edge)
- Minimal aspect ratio := $\max\{\frac{W}{H}, \frac{H}{W}\}$
- Minimal area
- Order preserving (edge curve to left child is $x$-decreasing, to right child is $x$-increasing)

1.2 How to Draw a Binary Tree

1.2.1 The easy way

Simply define a node’s $x$-coordinate to be its in-order number and its $y$-coordinate to be the negative of its depth.

This drawing technique has a few of the above mentioned properties but it’s not great with regards to symmetry, edge length, aspect ratio, and area. One particularly nice feature, however, is that the $x$-coordinate corresponds to binary search tree order.
1.3 The Tidy Way

Here is a simple recursive algorithm to produce a tidy drawing of a binary tree.

1. Construct the tidy drawing of the current node’s left subtree
2. Construct the tidy drawing of the current node’s right subtree
3. Shift the two subtree drawings to create a minimum gap of 2 between them
4. Shift them down by 1
5. Centre the current node between them horizontally

The tricky part of this algorithm lies in step 3. A naive implementation of this algorithm will follow the following recurrence relation:

\[ T(n) = T(l) + T(r) + \Theta(n) = \Theta(n^2) \]

where \( l \) is the number of nodes in the left subtree, \( r \) the number of nodes in the right subtree. The \( \Theta(n) \) comes from searching for the closest gap between the two trees. However, we only need to calculate the gap at levels that are occupied in both the left and right subtree. This takes time proportional to the minimum of the two subtree heights. By recording the contours in a relative fashion, we can merge the subtree contours to obtain the contour of the parent in the same amount of time. In Figure 3, the left (blue) subtree offsets by \((-1, -1)\) to move to the child level, then \((+1, +1)\) to move to the grandchild level, then \((-1, +1)\) to move to the great grandchild level. It is simple to determine the closest gap between the two subtrees incrementally and adjust it to 2.
Figure 3: Merging contours: \( L = [(-1, -1), (+1, +1), (-1, +1)] \) and \( R = [(-1, +1), (-1, -1)] \) to get \( T = [(-2, +2), (-1, +1), (+1, -1), (-1, -3)] \).

Figure 4: HV-Tree Drawing.

Rather than taking \( \Theta(n) \) time to find the closest gap, the running time will be proportional to the smallest subtree height, leading to the following recurrence relation:

\[
F(T) = F(L) + F(R) + \min\{h(L), h(R)\}
\]

where \( T \) is the current tree, \( L \) is its left subtree, \( R \) is its right subtree, and \( h(T) \) is the height of tree \( T \). The solution of the recurrence is \( F(T) = n(T) - h(T) \) where \( n(T) \) is the number of nodes in tree \( T \). An inductive proof is fairly straightforward (I omit the base case for brevity):

\[
F(T) = F(L) + F(R) + \min\{h(L), h(R)\} \quad \text{(by definition)}
\]
\[
= n(L) - h(L) + n(R) - h(R) + \min\{h(L), h(R)\} \quad \text{(by ind. hyp.)}
\]
\[
= n(L) + n(R) - \max\{h(L), h(R)\}
\]
\[
= (n(T) - 1) - (h(T) - 1) = n(T) - h(T)
\]

This results in a linear time algorithm for drawing, which is great, but it still could require \( \Omega(n^2) \) area in certain cases.

1.4 Towards Recursive Winding

1.4.1 HV-Drawing

An \textit{HV-Drawing} is a tree-drawing technique where child subtrees can be placed horizontally or vertically with respect to each other in the following manner.

For a horizontal arrangement, place the left subtree with a gap of 1 horizontally from the root of the right subtree, offset 1 below the root of the right subtree, and draw the parent 1 unit directly
above the root of the left subtree. For a vertical arrangement, place the left subtree with a gap of 1 vertically from the bottom of the right subtree, offset 1 to the left of the root of the right subtree, and draw the parent 1 unit directly to the left of the right subtree (see Figure 4).

1.4.2 Right-Heavy H-Drawing

A Right-Heavy H-Drawing is a simple technique for producing a small area HV-Drawing (though it always uses ‘H’): always put the subtree with the most nodes on the right. This drawing has several important properties:

- Height $\leq \log n$
- Width $\leq n - 1$
- Aspect ratio is bad: $\Omega(n/\log n)$ worst case

The $\log n$ height follows since every vertical edge leads to a subtree that is at most half the size of its parent’s subtree. Thus, at most $\log n$ edges can be traversed vertically.

1.5 Recursive Winding

We make the assumption that every node except the leaves has two children, so $n = 2^l - 1$. We can simply delete nodes from the drawing at the end to recover the original graph. Rearrange the graph so that $\text{leaves(left}(v)) \leq \text{leaves(right}(v))$ for every node $v$ in the graph. Note that the tree is no longer order preserving. Let $A$ be some parameter to be determined later:

- if $\text{leaves}(T) \leq A$, use a right-heavy H-Drawing
- otherwise
  - Choose $k$ such that $l_1 + \cdots + l_{k-1} < A$ and $l_1 + \cdots + l_{k-1} + l' \geq A$
  - Recursively draw $L$, $R$, and $T_1, \ldots, T_{k-1}$
  - Combine the drawings according to Figure 6.
Figure 6: Recursive Winding combination step.

Note that by construction \( l' \leq l'' \), and \( k \) is chosen so that \( l'' \leq l - A \). If we let \( H(l) \) be the height of a drawing with \( l \) leaves and \( W(l) \) the width, then the following recurrence relations define them:

\[
H(l) \leq \max \{ H(l') + H(l'') + \log_2 A + 3, l_{k-1} - 1 \}
\]
\[
W(l) \leq \max \{ W(l') + 1, W(l''), l_1 + \cdots + l_{k-2} + \log_2 l_{k-1} + 1 \}
\]

Let’s look at \( W \) first. Note that \( l_1 + \cdots + l_{k-2} < A \) by our choice of \( k \). Also, \( \log_2 l_{k-1} \leq \log_2 A \). Thus, \( W(l) \leq \max\{W(l') + 1, W(l''), A\} + \log_2 A + 1 \). Since \( l' \) and \( l'' \) are smaller than \( l - A \), the recurrence happens at most \( \frac{l}{A} \) times. Thus \( W(l) \in O\left(\frac{l}{A} \log_2 A + A\right) \). Looking at \( H \), the recurrence relation isn’t quite as obvious. The following lemma will make it easier:

**Lemma 1.** If \( A > 1 \) and \( f \) is a function that obeys:

- \( l \leq A \) implies \( f(l) \leq 1 \), and
- \( l > A \) implies \( f(l) \leq f(l') + f(l'') + 1 \) for some \( l' \leq l'' \leq l - A \) with \( l' + l'' \leq l \)

Then for all \( l > A \), \( f(l) < \frac{4l}{A} - 1 \).

This can be proved by induction on \( l \). The lemma directly applies to \( H \) and we get \( H(l) \in O\left(\frac{l}{A} \log_2 A + A\right) \). Thus \( W \) and \( H \) are both \( O\left(\frac{l}{A} \log_2 A + A\right) \). It turns out, moreover, that they are both \( \Theta\left(\frac{l}{A} \log_2 A + A\right) \). Therefore they are asymptotically equivalent and the aspect ratio is constant! Now we just have to choose \( A \) to minimize the area \( H \times W \), and it works out to be \( A = l \log_2 l \), and the area is thus \( O(n \log n) \).
1.6 Ring Drawings

This technique is more of a heuristic technique and not as much rigor and analysis can be applied, but it is certainly interesting. The algorithm is as follows:

1. let \( f(k) = 1 - \frac{(1-\sin \pi k)^2}{(1+\sin \pi k)^2} \)

\[ f(k) = 1 - \frac{\text{red area}}{\text{total area}} \]

2. sort children by their number of children

3. pick \( k \) so that the number of grandchildren attributed to the top \( k \) children \( \approx f(k) \) fraction of all grandchildren

4. recursively draw top \( k \) child subtrees

5. draw remaining child subtrees in the inner circle

6. rotate child drawings to minimize overlap

This allows for a well spaced and somewhat symmetrical representation of trees with high branching factors.

![Figure 7: A Ring drawing of a tree.](image)

1.7 Hyperbolic Drawings

Hyperbolic Drawings are another heuristic technique for drawing trees where the drawing is zoomed-in about the centre (initially the root) and nodes diminish in size the farther they are from the centre. If a viewer is allowed to move the centre, this has the effect of enhancing local/relevant nodes while pushing far away and less relevant nodes to the edges.
Figure 8: A Hyperbolic drawing of a tree.
References


