CS715: lecture 2

Tree Drawing Algorithms and Visualization Methods

Michael J. Dinneen

(Slightly modified version of Kai (Kevin) Xu slides.)
Tree Drawing Algorithm

- terminology
- layered drawing
- hv-drawing
- radial drawing
Terminology

• Tree:
  – acyclic connected graph

• Rooted tree
  – Root: a distinguished vertex in the tree
  – Usually treated as a directed graph: all edges oriented away from the root
  – Direct edge \( u \rightarrow v \): \( u \) is the parent of \( v \) and \( v \) is the child of \( u \)
  – Leaf: no child

• Ordered tree: rooted tree with an ordering (left-to-right) for the children of every vertex
Terminology

- Binary tree: rooted tree with every node has at most two children
  - Left and right child
  - One child, either left or right
- Subtree rooted at $v$: the subgraph induced by all vertices that have $v$ as their “ancestor”
  - Binary tree: left and right subtree
- Depth of a vertex: number of edges between it and the root
  - Height of a tree: maximum depth
terminology
layered drawing
hv-drawing
radial drawing
A rather simple layering algorithm

1. Placing vertex with depth \( i \) into layer \( L_i \)
   - Root in the top layer \( L_0 \).
   - \( y \)-coordinate decided.

2. Ordering vertices in each layer
   - Keep the left-right order of two vertices the same as their parents to avoid crossings.

3. Compute the \( x \)-coordinate.
   - Requirement 1: placing the parent in the horizontal span of its children (possibly in a central position).
   - Requirement 2: sometimes the ordering of the children is fixed.
Horizontal coordinate assignment

• Solution 1: using in-order traversal
  – Set the $x$-coordinate as vertex rank in in-order traversal

• Problems:
  – Much wider than necessary
  – Parent is not centered with respect to children
An improved layered tree drawing algorithm

- Divide and conquer
- Divide step:
  - recursively apply the algorithm to draw the left and right subtrees of T.
- Conquer step:
  - move the drawings of subtrees until their horizontal distance equals 2.
  - place the root \( r \) vertically one level above and horizontally half way between its children.
  - If there is only one child, place the root at horizontal distance 1 from the child.
Layered drawing: binary tree

• Two traversals
  – step 1. post-order traversal
    For each vertex $v$, recursively computes the horizontal displacement of the left & right children of $v$ with respect to $v$.
  – step 2. pre-order traversal
    Computes $x$-coordinates of the vertices by accumulating the displacements on the path from each vertex to the root.
Post-order Traversal

- **left (right) contour**: the sequence of vertices $v_i$ such that $v_i$ is the leftmost (rightmost) vertex of $T$ with depth $i$

- After we process $v$, we maintain the left and right contour of the subtree rooted at $v$ as a linked list.

- In conquer step, we need to follow the right contour of the left subtree and the left contour of the right subtree
Left and right contour of subtree

Compute the left and right contour of vertex $v$:

- scan the right contour of the left subtree ($T'$) and the left contour of the right subtree ($T''$)
- accumulate the displacements of the vertices on the left & right contour
- keep the max cumulative displacement at any depth

Three cases:

- case 1: $\text{height}(T') = \text{height}(T'')$
- case 2: $\text{height}(T') < \text{height}(T'')$
- case 3: $\text{height}(T') > \text{height}(T'')$
Left and right contour of subtree

- \( L(T) \) (\( R(T) \)): left (right) contour of the subtree \( T \) rooted at \( v \)
- case 1: \( \text{height}(T') = \text{height}(T'') \)
  - \( L(T) = L(T') + v \)
  - \( R(T) = R(T'') + v \)
- case 2: \( \text{height}(T') < \text{height}(T'') \)
  - \( R(T) = R(T'') + v \)
  - \( L(T) = v + L(T') + \{\text{part of } L(T'') \text{ starting from } w\} \)
    - \( h' \): depth of \( T' \)
    - \( w \): the vertex on \( L(T'') \) whose depth \( = h' + 1 \)
- case 3: \( \text{height}(T') > \text{height}(T'') \) : similar to case 2
Layered drawing

• Two traversals
  – step 1. post-order traversal
  – step 2. pre-order traversal
  • Computes $x$-coordinates of the vertices by accumulating the displacements on the path from each vertex to the root.
Time Complexity

• Pre-order traversal (step 2):
  – linear

• Post-order traversal (step 1):
  – Linear, but why?
    – it is necessary to travel down the contours of two subtrees \( T' \) and \( T'' \) only as far as the height of the subtree of lesser height
    – the time spent processing vertex \( v \) in the post-order traversal is proportional to the minimum heights of \( T' \) and \( T'' \)
    – The sum is no more than the number of vertices of the tree
    – Can be visualized by connecting vertices with same depth

• Hence, the algorithm runs in linear time
Drawing width and area

• Local horizontal compaction at each conquer step does not always compute a drawing of minimal width
  – can be solved in polynomial time using linear programming

• Area:
  – $O(n^2)$
Generalization

• generalization to rooted trees
  – reasonable drawing
  – root is placed at the average x-coordinates of its children
  – small imbalance problem:
    • The picture show the result when the algorithm works from left to right.
terminology
layered drawing
radial drawing
hv-drawing
Radial Drawing

- A variation of layered drawing
- Root at the origin
- Layers are concentric circles centered at the origin
- Usually draw each subtree in an annulus wedge $W$
Wedge angle

- Choose wedge angle to be proportional to the leave number in the subtree
- Problem: edge intersecting with level circle
Wedge angle

- To guarantee planarity, define convex subset $F$ of the wedge.
- The tangent to circle $c_i$ through $v$ meet circle $c_{i+1}$ at $a$ and $b$
- The unbounded region $F$ formed by the line segment $ab$ and the rays from origin through $a$ and $b$ is convex
- The final wedge angle is the lesser between the angle of $F$ and angle proportional to number of leaves.
Time and area

- Time:
  - Linear
- Area
  - Polynomial
  - Equal distance between circles
  - Tree height: $h$
  - Maximum child number: $d_M$
- Area:
  - $O(h^2d_M^2)$
  - First circle has perimeter as least $d_M$
  - It’s radius is $O(d_M)$
  - The radius of final circle is $O(hd_M)$
Radial Drawing

- Used for free trees (tree without a root)
  - Select a root minimize tree height
  - Can be found in linear time using simple recursive leaf pruning algorithm
  - One or two centers

- Variations:
  - choice of root,
  - radii of the circles,
  - how to determine the wedge angle
1. terminology
2. layered drawing
3. radial drawing
4. hv-drawing
HV-Drawing – binary tree

- hv-drawing of a binary tree $T$: straight-line grid drawing such that for each vertex $u$, a child of $u$ is either
  - horizontally aligned with and to the right of $u$, or vertically aligned with and below $u$
  - the bounding rectangles of the subtrees of $u$ do not intersect
- planar, straight-line, orthogonal, and downward
Divide & Conquer method

• Divide: recursively construct hv-drawings for the left & right subtrees
• Conquer: perform either
  – a horizontal combination or
  – a vertical combination
• The height & width are each at most $n-1$
Right-Heavy-HV-Tree-Draw

1. Recursively construct drawing of the left & right subtrees

2. Using only *horizontal* combination, place the subtree with the *largest* number of vertices to the *right* of the other one.
Right-Heavy-HV-Tree-Drawing

- hv-drawing (downward, planar, grid, straight-line and orthogonal)
- width is at most $n-1$
- height is at most $\log n$
  - The larger subtree is always placed to the right
  - The size of parent subtree is at least twice the size of vertical child subtree
- area $O(n\log n)$
Area aspect ratio

• **Right-Heavy-HV-Tree-Draw**
  - Good area bound, but bad aspect ratio

• **Better aspect ratio:**
  - use both vertical and horizontal combinations
  - Alternating the combination
    • Odd level: horizontal, even level: vertical
  - $O(n)$ area and constant aspect ratio
Optimization and extension

• It is possible to construct an hv-drawing of a binary tree that is optimal with respect to area or perimeter in $O(n^2)$ time.
  – use dynamic programming approach

• Right-Heavy-HV-Tree-Drawing can be extended to general rooted tree
  – downward, planar, grid, straight-line
  – area $O(n\log n)$
  – width is at most $n-1$
  – height is at most $\log n$
Tree Visualization

• While tree drawing algorithm is more theoretical, tree visualization is more applied.

• We will see examples of different tree visualization methods.
Indented Layout

- Places all items along vertically spaced rows
- Uses indentation to show parent child relationships
- Example: Windows explorer
- Problems:
  - Only showing part of the tree
  - Bad aspect ratio (not space efficient)
- But still the most popular one!?
Dendrogram

• Essentially a layered drawing
  – with bended orthogonal edges

• Layering are done according to the leaves:
  – All the leaves are on the same layer

• Now commonly used in bioinformatics to represent
  – The result of hierarchical clustering
  – Phylogenetic trees
  – More on this in the “biological networks” lecture
Balloon trees

• A variation of radial layout
• children are drawn in a circle centered at their parents.
• Effective on showing the tree structure
  – At the cost of node details
Hyperbolic Tree

• Simulate the distortion effect of fisheye lens
  – Enlarge the focus and shrink the rest
  – Focus+context
  – Interaction technique; can be combined with different layout.

• 3D hyperbolic tree:
  – projecting a graph on a sphere produces a similar distortion effect
  – This example also uses balloon tree drawing.
3D tree visualization - Cone tree

- Cone trees are a 3D extension of the 2D layered tree drawing method.
  - Parent at the tip of a cone, and its children spaced equally on the bottom circle of the cone
  - Either horizontal or vertical
- The extension to 3D does not necessarily means more information can be displayed
  - Occlusion problem
  - Couple with interaction is essential
  - More on this in the “graph visualization evaluation” lecture
Other 3D tree visualizations

• 3D poly-plane tree visualization
  – Put subtrees on planes
  – arrange these planes in 3D to reduce occlusion
  – In this example, layered drawing is used within each plane

• 3D layered tree
  – Only one cone
  – Layers are the parallel circles on the surface
  – Example: color indicate the layer
Space-filling methods - Treemap

- Treemap uses containment to show the hierarchy.
- It partitions the space recursively according to the size of subtrees.
- It is space-efficient compared to node-link diagrams.
- It is effective in showing the leaf nodes; on the other size, it is difficult to see the non-leave nodes.
Variations of treemap

• Cushion treemap
  – Use shading to help identify the levels in a treemap

• Voronoi treemap
  – Similar idea but uses voronoi diagram as partition
  – The space does not have to be rectangle.
Beamtree

- A variation of treemap in 3D.
- Using overlap instead of nesting to show the hierarchy
- 3D version: representing each node as a beam
- A bigger example
Space-filling tree layout

- Try to use as much screen space as possible
- Layout a tree according to the recursive partition of the screen space
  - The area allocated to a subtree is proportional to its size.
- A bigger example: 55000 nodes
  - Use all the screen space
  - Not very effective on showing the tree structure
Other space filling methods - Icicle Trees

• Edges implied by adjacency and spatial relationship.

• icicle tree in the infovis toolkit (jean-daniel fekete)
Information slice and Sunburst Diagrams

• Information slice
  – also a space-filling visualization method.
  – Radial version of icicle trees.
  – Node size is proportional to the angle swept by a node.

• Sunburst
  – With extra focus+context
  – Details are shown outside or inside the ring
Elastic hierarchies

- hybrid of node-link diagrams and treemaps
- Using node-link diagram inside a treemap produces lots of crossings
TreeViewer

- Visualizes trees in a form that closely resembles botanical trees
  - The root is the tree stem
  - Non-leave nodes are branches
  - Leave nodes are “bulbs” at the end of branches
  - Example: Unix home directory.
Collapsible Cylindrical Trees

- Telescope metaphor: A set of nested cylinders
  - A cylinder is constructed for the children of a node, and it has a smaller radius.
  - This cylinder is nested and hidden within the cylinder that contain the parent.
  - It can be pulled out to the right of the parent cylinder or collapsed as necessary.
- only one path of the hierarchy is visible at once
  - represented by a number of ever decreasing cylinders
- All cylinders of level 1 nodes are shown in a horizontal fashion, like being put on a stick.