Tree Drawing

Vincent La

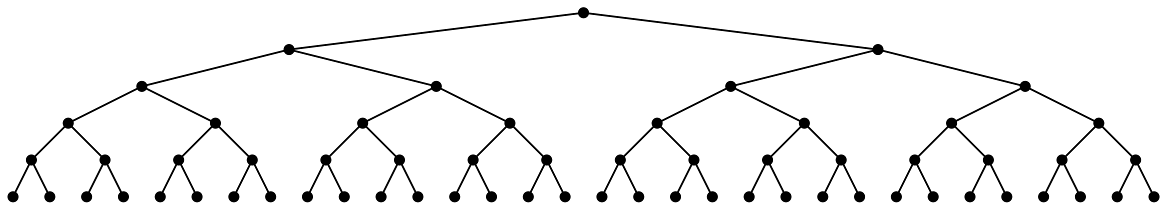
March 25, 2018

Motivation

Trees with at most two child nodes are used widely in computer science. The simplicity of their structure easily lends to mathematical analysis about algorithms operating on binary trees. Given the vast utility of this tree, many have tried to define algorithms which draw binary trees.

Reingold-Tillford (1981)

One classic algorithm used to layout binary trees is described by Reingold and Tillford.



Algorithm First, we calculate the displacements of the nodes relative to each other.

1. Base Case: Trivial
2. Apply this algorithm to subtrees via a postorder traversal
3. For each subtree, merge them horizontally such that they are two units apart horizontally

Pseudocode

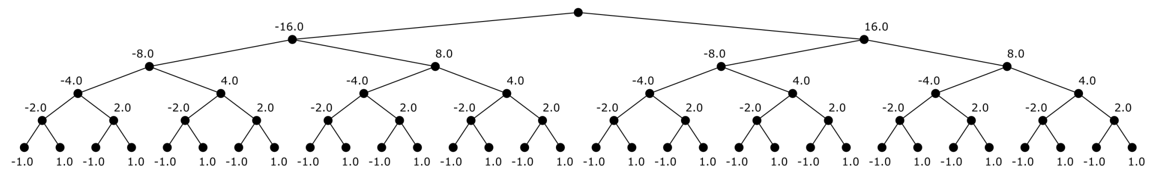
Distance Between

1. Keep a counter of the accumulated offsets for the left side ($left\_dist$) and right side ($right\_dist$). Also keep a counter for the maximum difference between the left and right side.
2. While there are still nodes on the left and right contour
   (a) Update $left\_dist$ and $right\_dist$
   (b) If $left\_dist + right\_dist > current\_dist$, update $current\_dist$

Algorithm Trace for Complete Binary Trees

The figures below show the displacements set by the algorithm for each node.

Figure 1: Algorithm trace for complete binary tree of height 6



N-Ary Tree Drawing

Code

```
// This file contains only the tree drawing algorithm itself

#include "tree.h"

void TreeNode::calculate_xy(const unsigned int depth, const double offset, const
    /** Calculate the x, y coordinates of each node via a preorder traversal
     */

    if (depth == 0) this->calculate_displacement();

    this->x = offset + (displacement * options.x_sep);
    this->y = depth * options.y_sep;

    if (left()) left()->calculate_xy(depth + 1, this->x, options);
    if (right()) right()->calculate_xy(depth + 1, this->x, options);
}

void TreeBase::calculate_displacement() {
    /** Calculate the displacement of each node via a postorder traversal */
    this->merge_subtrees(0);
}

double TreeBase::distance_between(TreeBase* left, TreeBase* right) {
    /* Return the minimum horizontal distance needed between two subtrees
     * such that they can be placed 2 units apart horizontally
     *
     * Precondition: All nodes except the top nodes have correct displacements s
     */

    double left_dist = 0, right_dist = 0, current_dist = 0;
    while (left && right) { // Terminate when either subtree runs out of height
        // Accumulate displacements
        if (left->displacement < 0) left_dist += abs(left->displacement);
        else right_dist += left->displacement;

        if (right->displacement < 0) left_dist += abs(right->displacement);
        else right_dist += right->displacement;

        // Update distance
        if ((right_dist + left_dist) > current_dist)
            current_dist = right_dist + left_dist;

        // Traverse right contour of left side
        if (left->right()) left = left->right();
        else left = left->left(); // If null, while terminates

        // Traverse left contour of right side
        if (right->left()) right = right->left();
        else right = right->right(); // If null, while terminates
    }

    return current_dist;
}

void TreeNode::merge_subtrees(double displacement) {
    /* "Merge" the subtrees of this node such that they have a horizontal separa
     *  by setting an appropriate displacement for this node
     *
     *  displacement: Default displacement value if this is a leaf node
     *  - Should be -1 for left nodes and 1 for right nodes
     */

    // Default displacement
    this->displacement = displacement;

    // Postorder traversal
    if (this->left()) this->left()->merge_subtrees(-1);
    if (this->right()) this->right()->merge_subtrees(1);

    // Merge subtrees (if they exist)
    if (this->left() && this->right()) {
        // Because by default, this node has displacement zero,
        // it will be centered over its children
        double subtree_separation = (this->distance_between(this->left(), this->
        this->left()->displacement = -subtree_separation;
        this->right()->displacement = subtree_separation;
    }
}
```