Orthogonal Straight Line Drawing of Trees

Sumi Y. Choi

Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the Degree of
Master of Science in the Department of Computer Science
at Brown University

June 3, 1999

Professor Roberto Tamassia
Advisor

Straight Line Drawing of Trees
Computational Geometry

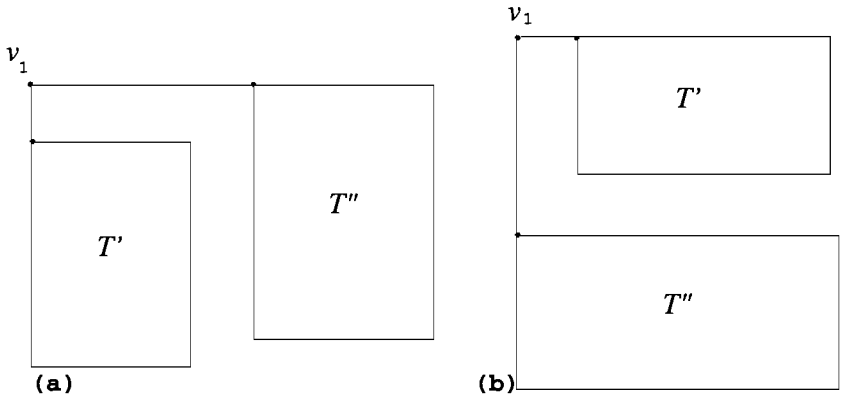
Sumi Y. Choi

May 26, 1999

This paper is written for the purpose of describing my ScM Project. It introduces a drawing algorithm for a tree type of graph. The drawing algorithm is the extension of the orthogonal straight line drawing[1], which is implemented in java and the javadoc documentation is attached in the appendix. While the orthogonal straight line drawing algorithm[1] is designed only for binary tree graph, This new algorithm is aimed for rooted tree graph with nodes of arbitrary number of children. The aspect ratio is considered as one of the important feature, so it becomes the target of analysis in the section 3, as it is in the orthogonal straight line drawing algorithm.

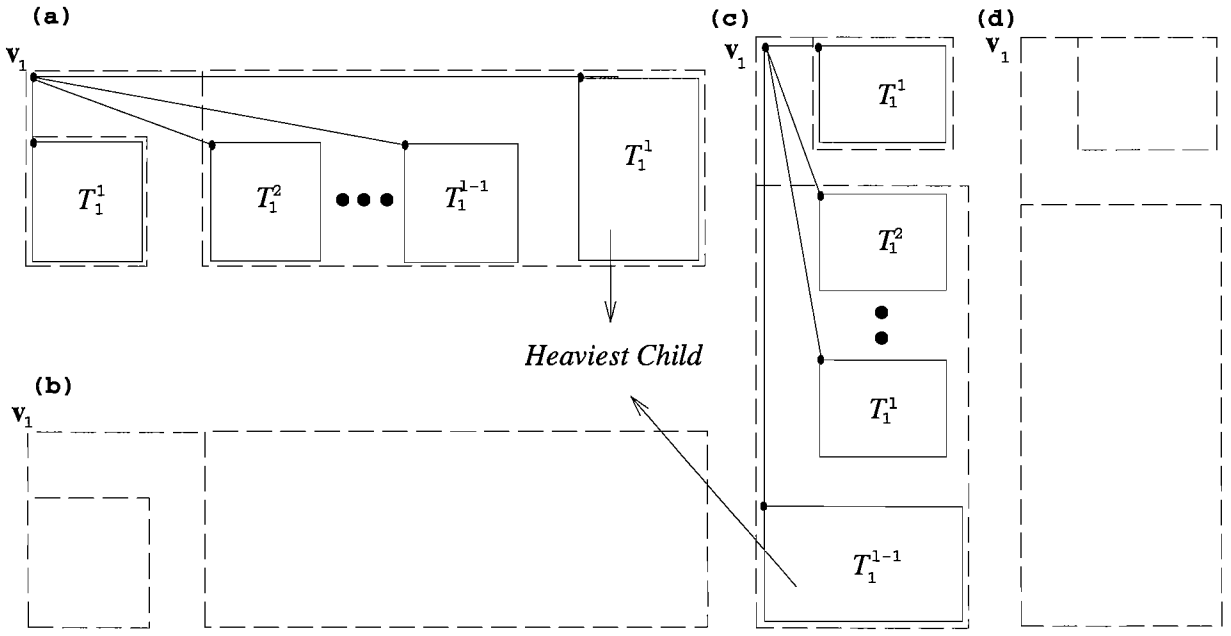
1 Base Drawing

To support the algorithm for trees with arbitrary children, the base case drawing algorithms need to be adjusted. Let's recall those algorithms for binary trees.
Assume that there is a binary tree and that under any of its nodes, the right child is heavier than the left child. In Figure 1, (a) is the drawing of height $O(\log n)$, width $O(n)$ and (b) is the drawing of height $O(n)$, and width $O(\log n)$.
Now, generalize the scheme for trees with arbitrary number of children.
Assume there is a tree and there are l number of children under v_1, the root.
Note the drawings for trees in Figure 2 for idea of this drawing scheme.
We assume that the tree is right weighted. Locally, if v_1 is right weighted, it means that the last subtree, T_1^l among the group of v_1's children, $T_1^1, ..., T_1^l$ is heavier than any of its sibling subtrees, i.e. it has the biggest number of leaves. Let's redefine 'right weighted' concept in this way for the whole tree.



(a) height O(logn), width O(n)
(b) height O(n), width O(logn)

Figure 1: Binary Drawings



(a) drawing of a tree with O(log n) height and O(n) width. (b) frame of (a)
(c) drawing of a tree with O(n) height and O(log n) width. (d) frame of (c)

Figure 2: General Case Tree Drawings

So, the same concept applies to every nodes of the tree.

Now, the drawing scheme is as simple as the one for binary trees.

Firstly, separate the first subtree, locate the root of the first subtree, T_1^1, and then draw its subtrees recursively. Secondly, consider the rest of the subtrees, separate the first subtree among the group of subtrees and follow the same procedure.

In Figure 2, (a) is a drawing of a tree with height $O(log_l n)$ and width $O(n)$, and (b) in Figure 2 shows a recursive structure for this drawing, similar to the binary tree drawing. With this drawing, we get non-orthogonal drawing, keeping the straight lining and grid coordinated properties.

In Figure 2, (c), (d) shows the drawing method of height $O(n)$, and width $O(log_l n)$. This second method follows almost the same procedure as the one described above, except the fact that determines the aspect ratio. As results of these tree drawings for a tree of n node with l children, we get the following results.

1. ***a planar upward straight-line drawing of a tree*** T ***with height at most*** $\lfloor \log_l n \rfloor$ ***and width*** $n-1$

2. ***a planar upward straight-line drawing of a tree*** T ***with width at most*** $\lfloor \log_l n \rfloor$ ***and height*** $n-1$

This drawing scheme can be constructed in $O(n)$ time.

Because I have been considering $l \geq 2$, the followings are also satisfied with the base drawings.

1. ***a planar upward straight-line drawing of a tree*** T ***with height at most*** $\lfloor \log_2 n \rfloor$ ***and width*** $n-1$

2. ***a planar upward straight-line drawing of a tree*** T ***with width at most*** $\lfloor \log_2 n \rfloor$ ***and height*** $n-1$

2 Upward Drawing

Next, I will extend the recursive winding scheme[1] for binary tree to a scheme for trees with nodes of arbitrary number of children. In the frames of Figure 2, the rectangles representing sub-drawings may not be a single

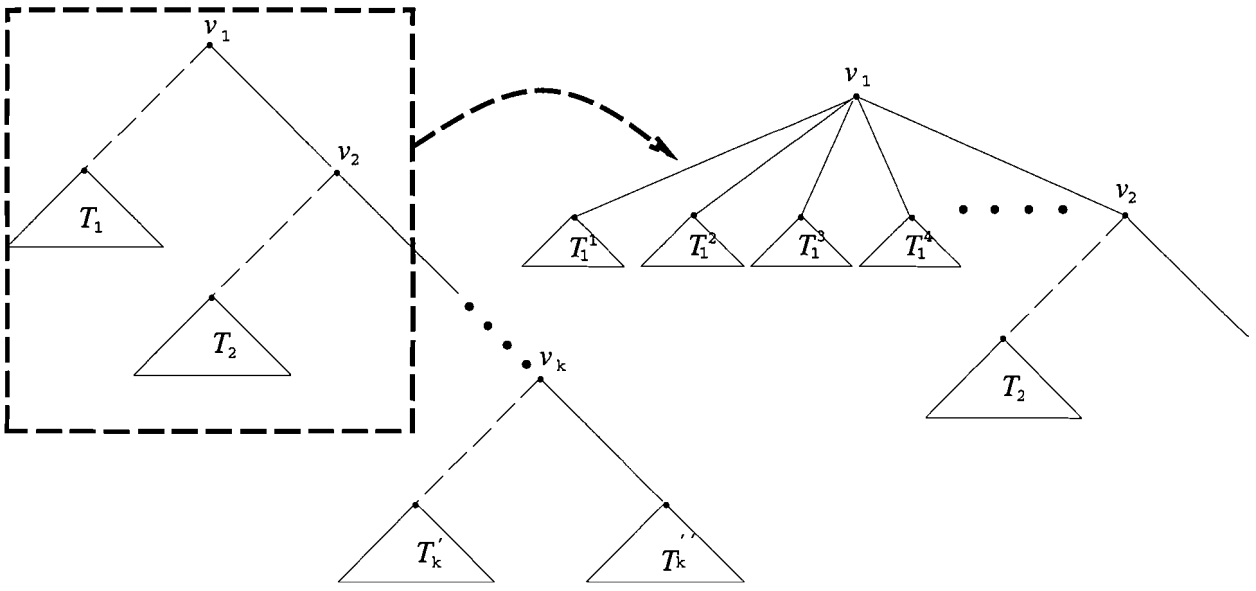


Figure 3: Right Weighted Tree

subtree, but a group of subtrees.
Consider a tree with nodes of arbitrary number of children. In recursive winding scheme[1], a binary tree is identified by its root node, the left and the right child. The right child is always heavy, or has more leaves than the left child does, with the two children defined recursively. Use the same idea to identify a general tree as follows.
A tree is identified by its root node, its heaviest child, and the rest of its children. The last element is a list of children except the heaviest one, or the one that has the biggest number of leaves. Each of the children is defined recursively. The Figure 3 explains this idea. A tree with more than two children is represented as if it were a binary tree by grouping subtrees together. It is shown on the left side of the Figure 3. The left child of each of the nodes v_1,...,v_k is denoted, $T_1, ..., T_k$ in this figure and each of those are again the groups of subtrees under each of the nodes, v_1,...,v_k. In other words, T_i is a group of subtrees, $T_i^1, ..., T_i^{l-1}$, assuming there are l number of children under v_i. In Figure 3, the detail inside the box of dotted line is shown on its right side.

Now, I would like to introduce the extended recursive winding scheme. The dotted line in Figure 4, which connect a node with the drawing of its

subtrees implies that the sub-drawing can consist of more than one subtrees and at the same time, the drawing can be non-orthogonal grid drawing.
As the binary case, fix a parameter $A > 1$ to be determined later.

If $n < A$, use the first base drawing method in section 1 for the whole tree. If not, follow the procedure described as below.
Note that the same notation is used for in [1], such as $N[v]$(number of leaves under v), T_m(the left child of a node v_m in Figure 3)... etc.
Now find the index k, such that $N[v_k] > n - A$ and $N[v_{k+1}] \leq n - A$.
In case of (a) in Figure 4, draw each of the subtrees of v_1 recursively. In case of (b) in Figure 4, draw the left child of v_1, T_1 first, using the first base drawing scheme. Then draw each of the subtrees of v_2, recursively. In case of (c) in Figure 4, draw the left child of each node, $v_1, ..., v_{k-2}$, i.e. T_1,...,T_{k-2}, using the first base drawing scheme. Then draw the left child of v_{k-1}, i.e. T_{k-2} using the second base drawing scheme. Again draw each of the subtrees of v_k, recursively toward the opposite direction.

3 Analysis

In this section, I would like to give the analysis of the drawing method given in the previous section. The analysis contains each of the height, the width and the time bound of a drawing.

Before proceeding to compute each of the $H(n)$, $W(n)$, and $T(n)$, I will first analyze the recurrence relation modified from [1].
Suppose $A > 1$, $m \geq 2$ and f is a function such that

- if $n \leq A$, then $f(n) \leq 1$; and
- if $n > A$, then $f(n) \leq f(n_1) + ... + f(n_l) + 1$ for some $n_1, ..., n_l \leq n - A$ with $n_1 + ... + n_l \leq n$ and $2 \leq l \leq m$.

Then

$$f(n) < (m+2)\frac{n}{A} - 1, \forall n \geq A \tag{1}$$

In the analysis of the recursive winding scheme, the variable A in the scheme will be plug into the variable A in this recurrence relation, the number of children under v_k, into the variable l. Also m will be taken from the maximum number of children that any node can have in the tree. In addition, the function $f(n)$ will be derived from each of the functions $H(n)$, $W(n)$ and $T(n)$ for the analysis.

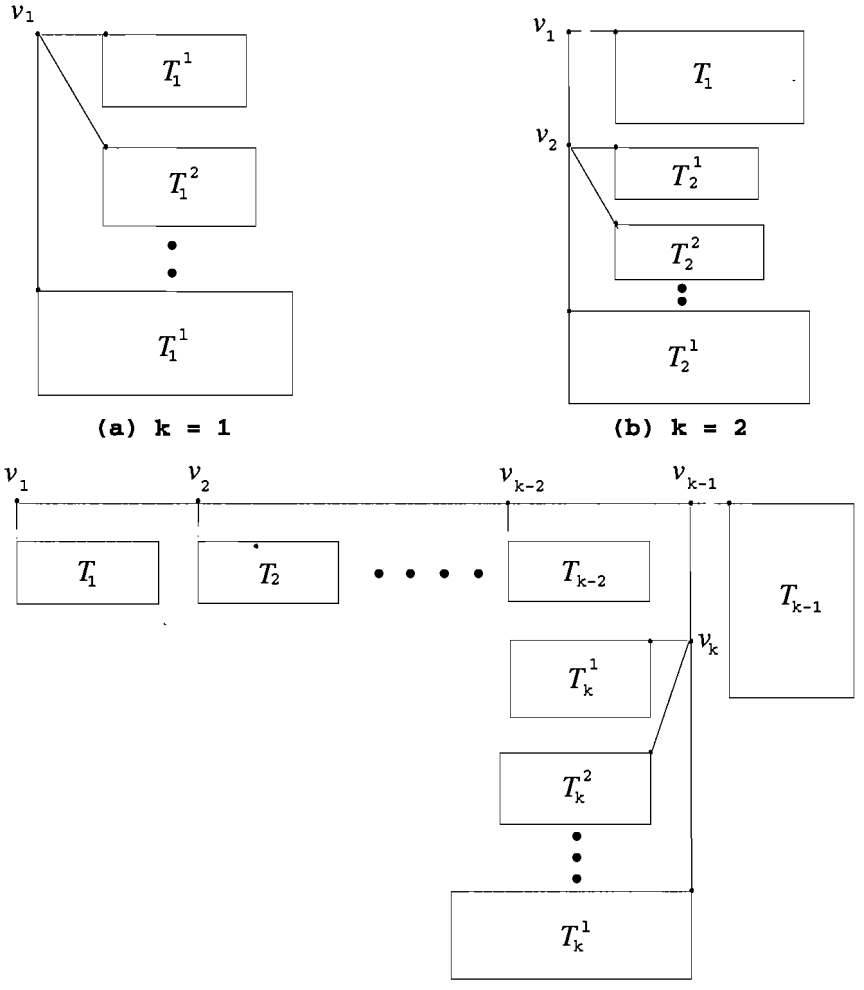


Figure 4: Upward Recursive Winding

Proof: The proof is by induction. Suppose the theorem is true for $n_1, ..., n_l$. If all $n_1, ...n_l < A$, then

$$f(n) \leq m + 1 < (m+2)\frac{n}{A} - 1$$

If $\exists$ non-empty set $I = \{j | 0 < j < l\}$ such that $\forall i \in I, n_i \leq A$, then assume $|I| = i$,

$$\begin{aligned}
f(n) &\leq i + \sum_{j \notin I} f(n_j) + 1 \\
&< i + \sum_{j \notin I} ((m+2)\frac{n_j}{A} - 1) + 1 \\
&= (m+2)\sum_{j \notin I} \frac{n_j}{A} - (l-i) + i + 1 \\
&= (m+2)\sum_{j \notin I} \frac{n_j}{A} - l + 2i + 1 \\
&< (m+2)\sum_{j \notin I} \frac{n_j}{A} - l + 2l + 1 \\
&< (m+2)\sum_{j \notin I} \frac{n_j}{A} + l + 1 \\
&< (m+2)\sum_{j \notin I} \frac{n_j}{A} + m + 2 - 1 \\
&< (m+2)(\sum_{j \notin I} \frac{n_j}{A} + 1) - 1 \\
&\leq (m+2)\frac{n}{A} - 1
\end{aligned}$$

Finally, if all $n_1, ...n_l > A$, then

$$\begin{aligned} f(n) &\leq \sum_{1\leq i\leq l} (\frac{f(n_i)}{A} - 1) + 1 \\ &< \sum_{1\leq i\leq l} ((m+2)\frac{n_i}{A} - 1) + 1 \\ &< \sum_{1\leq i\leq m} ((m+2)\frac{n_i}{A} - 1) + 1 \\ &< \sum_{1\leq i\leq l} (m+2)\frac{n_i}{A} + (m-1) \\ &< (m+2)\frac{n}{A} - 1 \end{aligned}$$

Now, let $H(n), W(n)$, and $T(n)$ denote the height, width, and construction time for drawing a tree.
Let $n_i = N[v_i]$(number of leaves of T_i), $n_k^1 = N[v_i^1], n_k^2 = N[v_i^2], ...$ and note the following properties:

$$n_1 + ... + n_{k-1} = n - N[v_k] < A \tag{2}$$

$$N[v_{k+1}] \leq n - A \tag{3}$$

In any case, the drawing for a tree of l number of children, can be made with the following bounds on the height, width, and construction time:

$$\begin{aligned} H(n) &= \max\{H(n_k^1) + ... + H(n_k^l) + l + 1 + \log A, n_{k-1} - 1\} \\ W(n) &= \max\{W(n_k^1) + 1, ..., W(n_k^{l-1}) + 1, W(n_k^l), n_1 + ... + n_{k-2}\} + \\ &\quad \log n_k + 1 \\ T(n) &= T(n_k^1) + ... + T(n_k^l) + O(n_1 + ... + n_{k-1} + 1) \end{aligned}$$

Then by property (2),

$$\begin{aligned} H(n) &= \max\{H(n_k^1) + ... + H(n_k^l) + l + O(\log A), A\} \\ W(n) &= \max\{W(n_k^1) + 1, ..., W(n_k^{l-1}) + 1, W(n_k^l), A\} + O(\log A) \\ T(n) &= T(n_k^1) + ... + T(n_k^l) + O(A) \end{aligned}$$

Now, we have the property (3), which implies that

$$n_k^1, ..., n_k^l \leq n - A \tag{4}$$

and

$$n_k^1 + ... + n_k^l < n. \tag{5}$$

Also, fix a variable m to be the maximum number of children in the target tree.
By (4), (5) and the recurrence relation (1), determine the boundaries of the drawings.
Firstly, consider the height $H(n)$.
If $A > H(n_k^1) + ... + H(n_k^l) + l + O(\log A)$,

$$H(n) = A \tag{6}$$

else $H(n) = H(n_k^1) + ... + H(n_k^l) + l + O(\log A)$.
Now, set $F(n) = \frac{H(n)+1}{O(\log A)}$.
If $n \leq A$,

$$\begin{aligned} H(n) &= O(\log A) \\ \Rightarrow F(n) &= \frac{H(n)+1}{O(\log A)} \leq 1. \end{aligned}$$

else if $n > A$,

$$\begin{aligned} H(n) + 1 &\leq (H(n_k^1) + 1) + ... + (H(n_k^l) + 1) + O(\log A) \\ \Rightarrow F(n) &\leq F(n_k^1) + ... + F(n_k^l) + 1 \end{aligned}$$

Finally by this result and the recurrence relation (1),

$$F(n) \leq (m+2)\frac{n}{A} - 1$$

, which in the function of $H(n)$, is

$$\begin{aligned} H(n) + 1 &\leq (m+2)(\frac{n}{A})O(\log A) \\ \Rightarrow H(n) &\leq (m+2)\lceil\frac{n}{A}\rceil O(\log A) - 1 \end{aligned}$$

$$H(n) = O(m\lceil\frac{n}{A}\rceil \log A) \tag{7}$$

By combining the results (6) and (7) from these cases, we get

$$\Rightarrow H(n) = O(m\lceil\frac{n}{A}\rceil \log A + A)$$

Now, set $F(n) = \frac{T(n)}{O(A)}$.
If $n \leq A$,

$$T(n) = O(n) \leq O(A)$$
$$\Rightarrow F(n) = \frac{T(n)}{O(A)} \leq 1.$$

else if $n > A$,

$$T(n) \leq T(n_k^1) + ... + T(n_k^l) + O(A)$$
$$\Rightarrow F(n) < F(n_k^1) + ... + F(n_k^l) + 1$$

By this result and the recurrence relation (1),

$$F(n) \leq (m+2)\frac{n}{A} - 1$$

, which in the function of $T(n)$, is

$$T(n) \leq (m+2)\lceil\frac{n}{A}\rceil O(A)$$

$$\Rightarrow T(n) = O(mn)$$

Different from $H(n)$ and $T(n)$, the recurrence relation is not used for $W(n)$.
If $A > \max\{W(n_k^1)+1, ..., W(n_k^{l-1})+1, W(n_k^l)\}$,

$$W(n) = A + O(\log A) \tag{8}$$

else

$$\begin{aligned} W(n) &= \max\{W(n_k^1)+1, ..., W(n_k^{l-1})+1, W(n_k^l)\} + O(\log A) \\ &\leq \max_{1\leq i\leq l}\{W(n_k^i)\} + 1 + O(\log A) \\ &< W(n-A) + 1 + O(\log A) \text{ (by (5))} \end{aligned}$$

This sequence can be solved as follows:

$$\begin{aligned} W(n) &\leq \lceil\frac{n}{A}\rceil(O(\log A) + 1) \\ &= O(\lceil\frac{n}{A}\rceil \log A) \end{aligned} \tag{9}$$

Finally, by combining (8) and (9), we get

$$W(n) = O(\lceil \frac{n}{A} \rceil \log A + A)$$

The obtained boundaries are:

$$\begin{aligned} H(n) &= O(m\lceil \frac{n}{A} \rceil \log A + A) \\ W(n) &= O(\lceil \frac{n}{A} \rceil \log A + A) \\ T(n) &= O(mn) \end{aligned} \tag{10}$$

We still have the unknown A parameter in these boundaries. Let's plug $A = \sqrt{n \log n}$, the same value as in the binary case of [1]. Then, we get a planar upward straight-line drawing of T with height, width and time as follows:

$$\begin{aligned} H(n) &= O(\frac{mn}{\sqrt{n \log n}} \log \sqrt{n \log n} + \sqrt{n \log n}) \\ &= O(\frac{mn}{\sqrt{n \log n}} (\log n + \log \log n) + \sqrt{n \log n}) \\ &= O(\frac{mn \log n}{\sqrt{n \log n}} + \sqrt{n \log n}) \\ &= O(m\sqrt{n \log n}) \\ W(n) &= O(\frac{n}{\sqrt{n \log n}} \log \sqrt{n \log n} + \sqrt{n \log n}) \\ &= O(\frac{n}{\sqrt{n \log n}} (\log n + \log \log n) + \sqrt{n \log n}) \\ &= O(\frac{n \log n}{\sqrt{n \log n}} + \sqrt{n \log n}) \\ &= O(\sqrt{n \log n}) \\ T(n) &= O(mn) \end{aligned}$$

The final result of boundaries are

$$\begin{aligned} H(n) &= O(m\sqrt{n \log n}) \\ W(n) &= O(\sqrt{n \log n}) \\ T(n) &= O(mn) \end{aligned} \tag{11}$$

We have defined m as the maximum number children under the internal nodes of the target tree. With the drawing scheme described ahead, we get

the aspect ratio $\frac{H(n)}{W(n)} = O(m)$. However, this aspect ratio can result in a very unbalanced drawing as m approaches n. The aspect ratio can even be O(n) in some worst cases.
In this reason, let's look at the results with $A = \sqrt{mn \log n}$.

$$
\begin{aligned}
H(n) &= O(\frac{mn}{\sqrt{mn \log n}} \log \sqrt{mn \log n} + \sqrt{mn \log n}) \\
&= O(\frac{\sqrt{mn}}{\sqrt{\log n}}(\log m + \log n + \log \log n) + \sqrt{mn \log n}) \\
&= O(\frac{\sqrt{mn}}{\sqrt{\log n}}(\log m + \log n) + \sqrt{mn \log n}) \\
&= O(\frac{\sqrt{mn}}{\sqrt{\log n}} \log m + \sqrt{mn \log n}) \\
&= O(\sqrt{mn \log n}) \\
W(n) &= O(\frac{n}{\sqrt{mn \log n}} \log \sqrt{mn \log n} + \sqrt{mn \log n}) \\
&= O(\frac{n}{\sqrt{mn \log n}}(\log m + \log n + \log \log n) + \sqrt{mn \log n}) \\
&= O(\frac{n}{\sqrt{mn \log n}}(\log m + \log n) + \sqrt{mn \log n}) \\
&= O(\frac{\log m}{\sqrt{m}} \frac{\sqrt{n}}{\sqrt{\log n}} + \frac{\sqrt{n \log n}}{\sqrt{m}} + \sqrt{mn \log n}) \\
&= O(\sqrt{mn \log n}) \\
T(n) &= O(mn)
\end{aligned}
$$

Now, we get the height and the width of $O(\sqrt{mn \log n})$.

4 Experiments

To test this drawing scheme, the experiments are performed with several different types of the m value, which is the maximum number of children of the target tree. In the figure 5, there are results from the experiments for 5 types of m value. The first four examples are the upward and non-upward drawings of the cases of $m = 2, 3, 4, 10$, and $m = \sqrt{n}$, where n is the number of leaves. The first row in the results is in the binary cases, which shows the best and constant aspect ratio among the group of examples. As the m value grows bigger the data of aspect ratio gets scattered. It occurs notably when $m = 10$ on the fourth row in the results. There is one more case when $m = \sqrt{n}$, where n is the number of nodes. It means that the tree has very

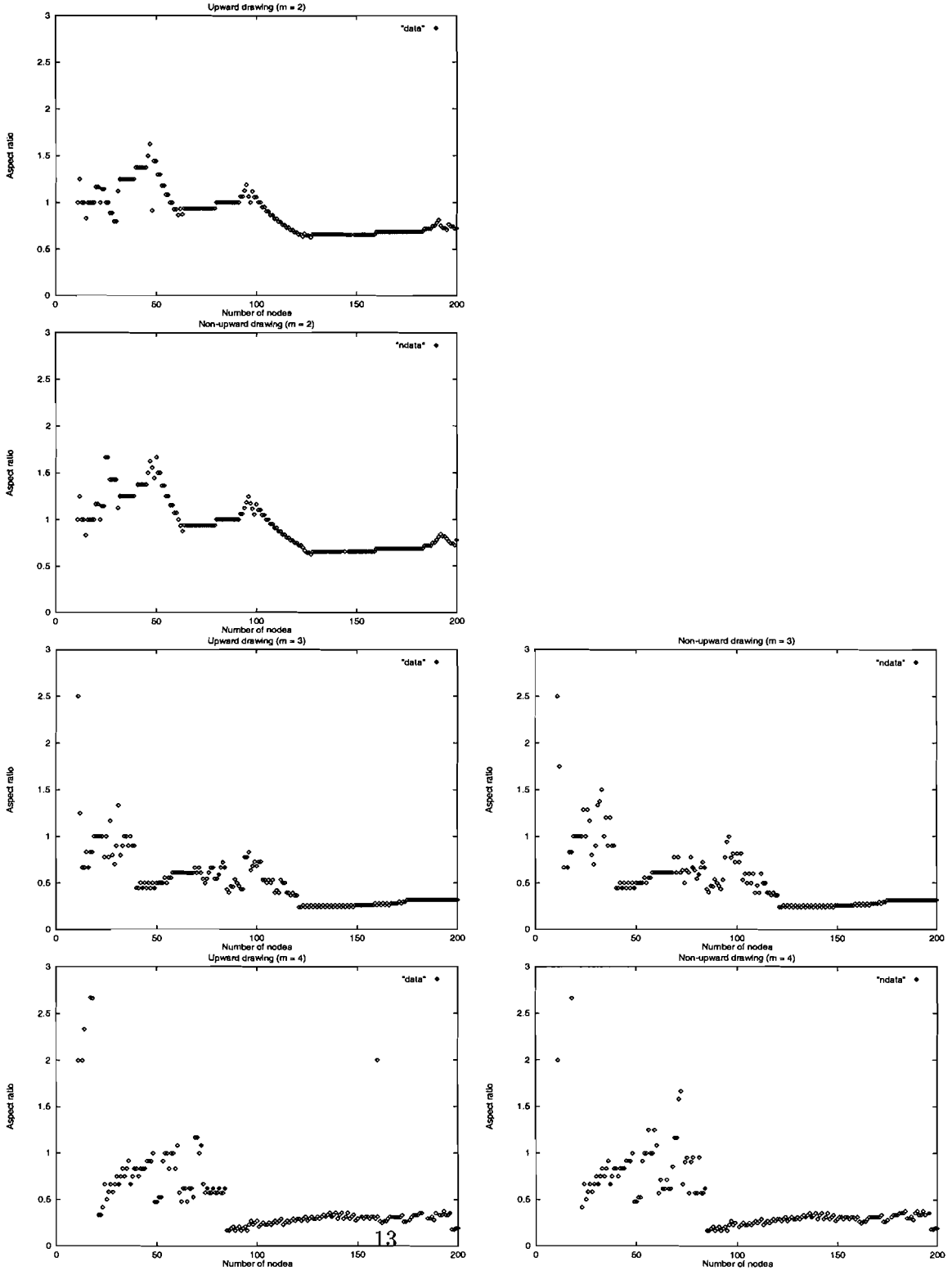


Figure 5: Experimental result, m = 2,3,4

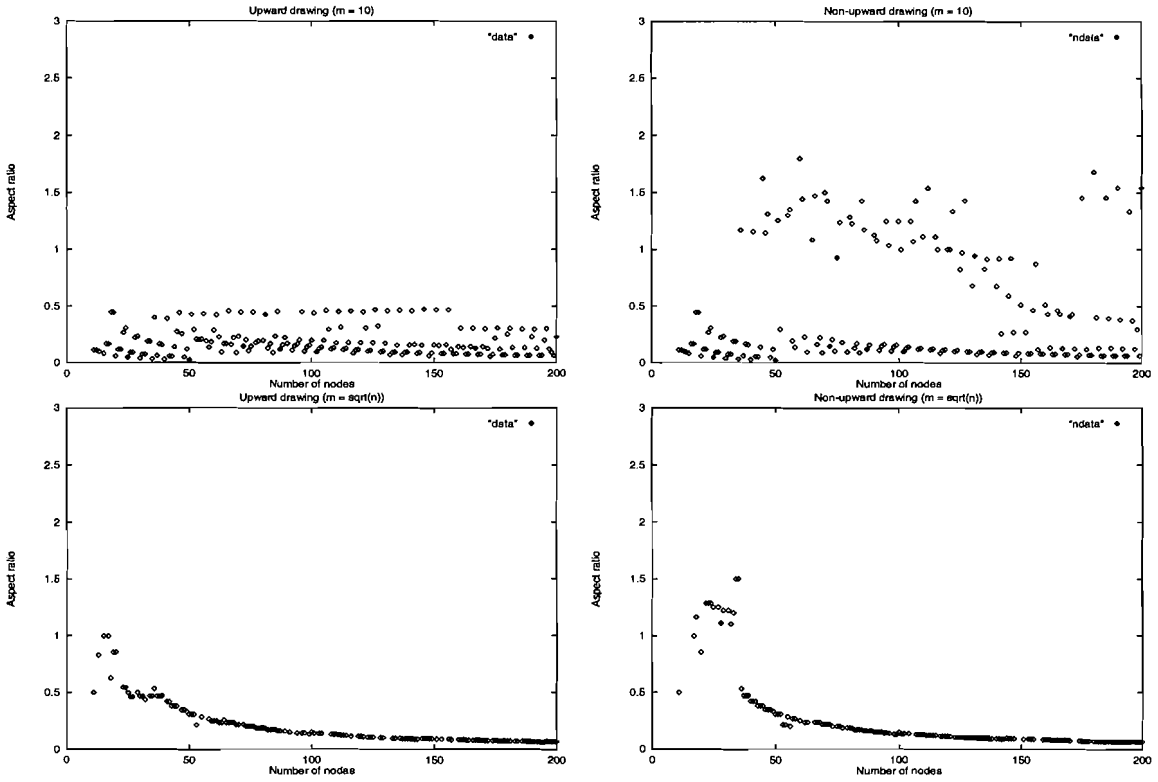


Figure 6: Experimental result, m = 10

huge number of children when the tree is relatively big. The aspect ratio in this case shown in the example gets worse as the tree gets bigger. In addition, the drawing of a subtree of big number of branches is not visible with human eyes either.

5 Conclusion

We have investigated the extension to the straight-line orthogonal tree drawings. The object class of this drawing scheme has expanded so that it contains not only binary trees, but also non-binary trees. This scheme renders non-orthogonal straight-line drawing. The analysis of the drawing shows that the height and the width are m times those of the drawing for binary trees in [1]. However, there are some types of trees that do not follow this result, which has been introduced in the experimental section. It might be useful if there is some scheme for trees with big number of m value.

References

[1] T. Chan, M.T.Goodrich, S.R.Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings.

A Javadoc documentation

As a part of the work, I have implemented the orthogonal straight-line drawing of binary tree in Java. The attached is the javadoc documentation for the implementation.

Class OrthoTreeDrawing

```
java.lang.Object
   |
   +---OrthoTreeDrawing
```

---

public class **OrthoTreeDrawing**
extends Object
implements GraphDrawing

Variable Index

- **graph_**
  Input tree
- **nleaves_**
  Hashtable containing the number of end-leaves per each node
- **nnodes_**
  Hashtable containing the number of total nodes per each subtree
- **NULL_POINT**
  Empty point symbol 1
- **NULL_POINT2**
  Empty point symbol 2
- **NULL_SEG**
  Empty segment symbol
- **tree_**
  Input tree
- **x_dir_**
  The horizontal direction of the drawing
- **y_dir_**
  The vertical direction of the drawing

Constructor Index

- **OrthoTreeDrawing**(InspectableTree)
  Constructor

Method Index

- **adjustHeight**(BinaryTree, Position, int)
  adjust the subtree under the root by by_y amount on the y-axis
- **adjustSubtree**(BinaryTree, Position, int)
  adjust the subtree under the root by by_x amount on the x-axis
- **balanceBinaryTree**(BinaryTree, Position)

recursively build a balanced binary tree
- **beautifyHeight**(BinaryTree, int, int, int, Position)
  Adjust the height of the subtree under the current position
- **beautifySubtree**(BinaryTree, int, int, int, Position, Position)
  Adjust the position of the subtree under the current position
- **binaryTree**(Position)
  check if the target tree is binary tree
- **buildBalancedBinaryTree**(InspectableTree)
  build a balanced binary tree(left subtree has less leaves than the right subtree)
- **clear**()
  Remove any decorations the algorithm may have attached to the graph.
- **computeK**(BinaryTree, Position, double, int)
  compute the index k from the orthogonal straight line drawing
- **computeNumLeaves**(BinaryTree, Position)
  Compute the number of leaves under the current position
- **computeNumNodes**(BinaryTree, Position)
  Computes the number nodes under the current position
- **copyInspectableGraphToBinaryTree**(BinaryTree, Position, Vertex)
  copy the inspectable graph back to binary tree
- **execute**()
  Computes the orthogonal straight line drawing of the given binary tree
- **geomObject2D**(Edge)
- **geomObject2D**(Vertex)
- **getK**(BinaryTree, Position)
  retrieve the index k
- **graph**()
- **hasGeomObject2D**(Edge)
- **hasGeomObject2D**(Vertex)
- **initialize**(InspectableTree)
  initialize the local variable according to the target tree
- **initPosition**(BinaryTree, Position)
  initialize the poitions of the target tree
- **makeInspectableGraphFromBinaryTree**(BinaryTree, Position, Vertex)
  get the inspectable graph from a binary tree
- **maxBoundary**(int, int)
  Relative maximum boundary(depends on the direction of the subtree drawn)
- **run**(BinaryTree)
  Run the algorithm
- **run_u_algo**(BinaryTree, Position, int, int)
  Run the original orthogonal straight line drawing
- **runK**(BinaryTree, Position, int, int)
  Run the algorithm for the case of K > 2
- **runT1**(BinaryTree, Position, int, int)
  Run the drawing algorithm for binary tree with O(logn) height and O(n) width
- **runT2**(BinaryTree, Position, int, int)
  Run the drawing algorithm for binary tree with O(logn) width and O(n) height
- **setAllEdges**(GeomObject2D)
- **setAllVertices**(GeomObject2D)
- **setGeomObject2D**(Edge, GeomObject2D)

- **setGeomObject2D**(Vertex, GeomObject2D)
- **setInspectableTree**(InspectableTree)
  set the object tree
- **transformSubtree**(BinaryTree, Position, int, int)
  transform the subtree under the root by (bx, by) on each axes

Variables

- **tree_**

```
private InspectableTree tree_
```

  Input tree

- **graph_**

```
private OnDemandGraph graph_
```

  Input tree

- **nleaves_**

```
private Hashtable nleaves_
```

  Hashtable containing the number of end-leaves per each node

- **nnodes_**

```
private Hashtable nnodes_
```

  Hashtable containing the number of total nodes per each subtree

- **x_dir_**

```
private int x_dir_
```

  The horizontal direction of the drawing

- **y_dir_**

```
private int y_dir_
```

  The vertical direction of the drawing

- **NULL_POINT**

```
private static final IntPoint2D NULL_POINT
```

  Empty point symbol 1

- **NULL_POINT2**

```
private static final IntPoint2D NULL_POINT2
```

Empty point symbol 2

- **NULL_SEG**

```
private static final IntSegment2D NULL_SEG
```

Empty segment symbol

Constructors

- **OrthoTreeDrawing**

```
public OrthoTreeDrawing(InspectableTree t)
```

Constructor

Parameters:
t - the object tree to be drawn by the algorithm

Methods

- **execute**

```
public void execute()
```

Computes the orthogonal straight line drawing of the given binary tree

- **clear**

```
public void clear()
```

Remove any decorations the algorithm may have attached to the graph.

- **graph**

```
public InspectableGraph graph()
```

Returns:
InspectableGraph the result graph of a inspectable graph based on the orthogonal straight line drawing

- **geomObject2D**

```
public GeomObject2D geomObject2D(Vertex v) throws InvalidPositionException
```

See Also:
geomObject2D

- **geomObject2D**

```
public GeomObject2D geomObject2D(Edge e) throws InvalidPositionException
```

See Also:
geomObject2D

- **hasGeomObject2D**

```
public boolean hasGeomObject2D(Vertex v) throws InvalidPositionException
```

See Also:
hasGeomObject2D

- **hasGeomObject2D**

```
public boolean hasGeomObject2D(Edge e) throws InvalidPositionException
```

See Also:
hasGeomObject2D

- **setGeomObject2D**

```
public void setGeomObject2D(Vertex v,
                            GeomObject2D g) throws InvalidPositionException
```

See Also:
setGeomObject2D

- **setGeomObject2D**

```
public void setGeomObject2D(Edge e,
                            GeomObject2D g) throws InvalidPositionException
```

See Also:
setGeomObject2D

- **setAllVertices**

```
public void setAllVertices(GeomObject2D g)
```

See Also:
setAllVertices

- **setAllEdges**

```
public void setAllEdges(GeomObject2D g)
```

- **beautifySubtree**

```
private int beautifySubtree(BinaryTree btree,
                            int initx,
                            int leftbound,
                            int rightbound,
                            Position p,
                            Position parent)
```

Adjust the position of the subtree under the current position

Parameters:

btree - binarytree
initx - initial x coordinate
leftbound - left boundary
rightbound - right boundary
p - position of the current node
parent - parent of the current node

- **beautifyHeight**

```
private int beautifyHeight(BinaryTree btree,
                           int inity,
                           int yoffset,
                           int ybound,
                           Position root)
```

Adjust the height of the subtree under the current position

Parameters:

btree - binarytree
inity - initial y coordinate
yoffset - the amount to be translated
ybound - height boundary
root - position of the current node

- **maxBoundary**

```
private int maxBoundary(int x1,
                        int x2)
```

Relative maximum boundary(depends on the direction of the subtree drawn)

Parameters:

x1 - first boundary
x2 - second boundary

- **runK**

```
private Point runK(BinaryTree btree,
                   Position p,
                   int x,
```

```
                    int y)
```

Run the algorithm for the case of K > 2

Parameters:

btree - binarytree
p - current position
x - initial x coordinate
y - initial y coordinate

- **runT1**

```
private Point runT1(BinaryTree btree,
                    Position p,
                    int x,
                    int y)
```

Run the drawing algorithm for binary tree with O(logn) height and O(n) width

Parameters:

btree - binarytree
p - current position
x - initial x coordinate
y - initial y coordinate

- **runT2**

```
private Point runT2(BinaryTree btree,
                    Position p,
                    int x,
                    int y)
```

Run the drawing algorithm for binary tree with O(logn) width and O(n) height

Parameters:

btree - binarytree
p - current position
x - initial x coordinate
y - initial y coordinate