Algorithms for graph visualization
Divide and Conquer - Tree Layouts
Applications

Decision tree analysis for prediction of outcome after traumatic brain injury

*Nature Reviews Neurology*
Applications

Level-based layout

Decision tree analysis for prediction of outcome after traumatic brain injury

_Nature Reviews Neurology_
Applications

Chart to aid students in shaping geographical questions by Gaultier, 1821
Applications

Chart to aid students in shaping geographical questions by Gaultier, 1821

X-MEN FAMILY TREE

Chart of the X-Men family tree.
An unrooted phylogenetic tree for myosin, a superfamily of proteins. "A myosin family tree“ *Journal of Cell Science*
Applications

Radial layout

An unrooted phylogenetic tree for myosin, a superfamily of proteins. 
"A myosin family tree“ Journal of Cell Science
Applications

Flare Visualization Toolkit code structure by Heer, Bostock and Ogievetsky, 2010

Greek Myth Family by Ribbecca, 2011
Cons cell diagram in LISP. 

*Cons*(constructs) are memory objects which hold two values or pointers to values.

![Diagram of cons cells of the simple tree.](http://gajon.org/)
Cons cell diagram in LISP. Cons (constructs) are memory objects which hold two values or pointers to values.

![Cons cell diagram](image)

**Figure 3:** Diagram of cons cells of the simple tree.

http://gajon.org/

**HV-layout (Horizontal-Vertical)**
Overview

- Applications with tree visualization
- Layered tree drawing algorithm
- H(horizontal) V(vertical) tree drawing algorithm
- Radial tree drawing algorithm
- Other visualization styles
Basic Definitions

- Tree - connected graph without cycles
- Binary tree
Basic Definitions

- Tree - connected graph without cycles
- Binary tree

Tree traversals

Root of the tree

left subtree $T_l(v)$

right subtree $T_r(v)$
Basic Definitions

- Tree - connected graph without cycles
- Binary tree

Tree traversals

Depth-first search
Basic Definitions

- Tree - connected graph without cycles
- Binary tree

Tree traversals

Depth-first search
- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)
Basic Definitions

- Tree - connected graph without cycles
- Binary tree

Tree traversals

Depth-first search
- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)

Breadth-first search
Basic Definitions

- Tree - connected graph without cycles
- Binary tree

Tree traversals

Depth-first search
- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)

Breadth-first search
- Assignes vertices to levels corresponding to depth
Basic Definitions

- Tree - connected graph without cycles
- Binary tree

Tree traversals

Depth-first search
- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)

Breadth-first search
- Assignes vertices to levels corresponding to depth

Isomorphism (of ordered trees)
- Simple
- Axial
Given: A rooted binary tree
Drawing of a Tree

**Given:** A rooted binary tree

**Question:** How would we draw it?
Drawing of a Tree

**Given:** A rooted binary tree

**Question:** How would we draw it?
Drawing of a Tree

**Given:** A rooted binary tree

**Question:** How would we draw it?
Given: A rooted binary tree
Question: How would we draw it?
Drawing of a Tree

**Given:** A rooted binary tree

**Question:** How would we draw it?

Vertices are mapped to levels
**Given:** A rooted binary tree  
**Question:** How would we draw it?

- Vertices are mapped to levels
Given: A rooted binary tree
Question: How would we draw it?

Vertices are mapped to levels
**Drawing of a Tree**

**Given:** A rooted binary tree

**Question:** How would we draw it?

- Vertices are mapped to levels
- Isomorphic trees are drawn similarly
**Given:** A rooted binary tree  
**Question:** How would we draw it?

- Vertices are mapped to levels
- Isomorphic trees are drawn similarly
**Given:** A rooted binary tree

**Question:** How would we draw it?

- Vertices are mapped to levels
- Isomorphic trees are drawn similarly
- Parent is centered wrt the children
Level-based Layout

**Algorithm Outline:**

**Input:** A binary tree

**Output:** A leveled drawing of $T$

**Base case:** A single vertex

**Divide:** Recursively apply the algorithm to draw the left and the right subtrees of $T$

**Conquer:**
Level-based Layout

Algorithm Outline:
Input: A binary tree
Output: A leveled drawing of T

Base case: A single vertex
Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:
Level-based Layout

Algorithm Outline:
Input: A binary tree
Output: A leveled drawing of T

Base case: A single vertex
Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:
Level-based Layout

Algorithm Outline:
Input: A binary tree
Output: A leveled drawing of $T$

Base case: A single vertex
Divide: Recursively apply the algorithm to draw the left and the right subtrees of $T$

Conquer:
Level-based Layout

Algorithm Outline:
Input: A binary tree
Output: A leveled drawing of T

Base case: A single vertex
Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:
Level-based Layout

Algorithm Outline:
**Input:** A binary tree
**Output:** A leveled drawing of T

**Base case:** A single vertex

**Divide:** Recursively apply the algorithm to draw the left and the right subtrees of T

**Conquer:**

Some agreed distance
Level-based Layout

Algorithm Outline:

**Input:** A binary tree

**Output:** A leveled drawing of $T$

**Base case:** A single vertex

**Divide:** Recursively apply the algorithm to draw the left and the right subtrees of $T$

**Conquer:**

Some agreed distance
Level-based Layout

Algorithm Outline:
Input: A binary tree
Output: A leveled drawing of T

Base case: A single vertex
Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer: Parent is centered wrp to children
Some agreed distance
Level-based Layout

Implementation Details (postorder and preorder traversals)

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

\[ T_l(v) \quad T_r(v) \]
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex \( v \) compute horizontal displacement of the left and the right child

- Assume at each vertex \( u \) (below \( v \)) we have stored the left and the right boundary of the subtree \( T(u) \)
- “Summ up” the horizontal displacements of the right boundary of \( T_l(v) \) and the left boundary of \( T_r(v) \) to obtain the displ. of the children of \( v \)
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- “Summ up” the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of $v$
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

1. Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
2. “Summ up” the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of $v$
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex \( v \) compute horizontal displacement of the left and the right child

- Assume at each vertex \( u \) (below \( v \)) we have stored the left and the right boundary of the subtree \( T(u) \)
- “Summ up” the horizontal displacements of the right boundary of \( T_l(v) \) and the left boundary of \( T_r(v) \) to obtain the displ. of the children of \( v \)
- Store at \( v \) the left and the right boundaries of \( T(v) \)
Level-based Layout

Implementation Details (postorder and preorder traversals)

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

**Preorder traversal:** Compute $x$- and $y$-coordinates.
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child.

Preorder traversal: Compute $x$- and $y$-coordinates.
Level-based Layout

Implementation Details (postorder and preorder traversals)

**Postorder traversal:** For each vertex \( v \) compute horizontal displacement of the left and the right child

**Preorder traversal:** Compute x- and y-coordinates.

\[
T_l(v) = (-2, -1), (0, 0), T_r(v) = (-1, +1)
\]
Implementation Details (postorder and preorder traversals)

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

**Preorder traversal:** Compute x- and y-coordinates.

![Diagram showing implementation details of level-based layout]
Level-based Layout

Time Complexity

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.
Level-based Layout

**Time Complexity**

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.
Level-based Layout

Time Complexity

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_{l}(v)$ and the left boundary of $T_{r}(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.
Level-based Layout

Time Complexity

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.
**Level-based Layout**

**Time Complexity**

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.
Level-based Layout

Time Complexity

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.
Level-based Layout

**Time Complexity**

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.
Level-based Layout

**Time Complexity**

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.
Level-based Layout

Time Complexity

**Postorder traversal:** For each vertex \( v \) compute horizontal displacement of the left and the right child

- Assume at each vertex \( u \) (below \( v \)) we have stored the left and the right boundary of the subtree \( T(u) \)
- Summ up the horizontal displacements of the right boundary of \( T_l(v) \) and the left boundary of \( T_r(v) \)
- Store at \( v \) the left and the right boundaries of \( T(v) \)

**Preorder traversal:** Compute x- and y-coordinates.
**Time Complexity**

**Postorder traversal:** For each vertex \( v \) compute horizontal displacement of the left and the right child

- Assume at each vertex \( u \) (below \( v \)) we have stored the left and the right boundary of the subtree \( T(u) \)
- Summ up the horizontal displacements of the right boundary of \( T_l(v) \) and the left boundary of \( T_r(v) \)
- Store at \( v \) the left and the right boundaries of \( T(v) \)

**Preorder traversal:** Compute x- and y-coordinates.
Level-based Layout

Time Complexity

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.

To compute the displacement: constant number of operations at each vertex
Level-based Layout

**Time Complexity**

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.

- To compute the displacement: constant number of operations at each vertex

\[ O(n) \]
Level-based Layout

Time Complexity

**Postorder traversal:** For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

**Preorder traversal:** Compute x- and y-coordinates.

- To compute the displacement: constant number of operations at each vertex
Level-based Layout

Time Complexity

Postorder traversal: For each vertex $v$ compute horizontal displacement of the left and the right child

- Assume at each vertex $u$ (below $v$) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at $v$ the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.

To compute the displacement: constant number of operations at each vertex
Level-based Layout

Theorem (Reingold & Tilford)

Let $T$ be a binary tree with $n$ vertices. Algorithm (R & T) constructs a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar and straight-line
- $\forall v \in T$ y-coordinate of $v$ is $-\text{depth}(v)$
- Vertical and horizontal distance is at least 1
- Area of $\Gamma$ is
**Level-based Layout**

**Theorem (Reingold & Tilford)**

Let $T$ be a binary tree with $n$ vertices. Algorithm (R & T) constructs a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar and straight-line
- $\forall v \in T$ y-coordinate of $v$ is $-\text{depth}(v)$
- Vertical and horizontal distance is at least 1
- Area of $\Gamma$ is $O(n^2)$
Level-based Layout

Theorem (Reingold & Tilford)

Let $T$ be a binary tree with $n$ vertices. Algorithm (R & T) constructs a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar and straight-line
- $\forall v \in T$ y-coordinate of $v$ is $-\text{depth}(v)$
- Vertical and horizontal distance is at least 1
- Area of $\Gamma$ is $O(n^2)$
- Each vertex is centered with respect to its children
Level-based Layout

Theorem (Reingold & Tilford)

Let $T$ be a binary tree with $n$ vertices. Algorithm (R & T) constructs a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar and straight-line
- $\forall v \in T$ y-coordinate of $v$ is $-\text{depth}(v)$
- Vertical and horizontal distance is at least 1
- Area of $\Gamma$ is $O(n^2)$
- Each vertex is centered with respect to its children
- Simply isomorphic subtrees have congruent (coincident) drawing, up to translation
- Axially isomorphic trees have congruent drawing, up to translation and reflection around y-axis
Level-based Layout

- The presented algorithm tries to minimize width
Level-based Layout

- The presented algorithm tries to minimize width
Level-based Layout

- The presented algorithm tries to minimize width
- Does not achieve that!
Level-based Layout

- The presented algorithm tries to minimize width.
- Does not achieve that!
- Divide-and-conquer strategy cannot achieve optimal width.
The presented algorithm tries to minimize width

Does not achieve that!

Divide-and-conquer strategy cannot achieve optimal width

Drawing with min width and properties of our algorithm can be constructed by an LP
Level-based Layout

- The presented algorithm tries to minimize width
- Does not achieve that!
- Divide-and-conquer strategy cannot achieve optimal width

- Drawing with min width and properties of our algorithm can be constructed by an LP
- If integer coordinates are required, then it is NP-hard
Level-based Layout - General trees

Algorithm Outline:

Input: A rooted tree
Output: A level-based drawing of \( T \)

Base case: A single vertex

Divide: Assume that \( T \) has subtrees \( T_1, \ldots, T_m \). Draw each \( T_i \) recursively.

Conquer:
Algorithm Outline:

Input: A rooted tree
Output: A level-based drawing of $T$

Base case: A single vertex

Divide: Assume that $T$ has subtrees $T_1, \ldots, T_m$. Draw each $T_i$ recursively.

Conquer: For $i = 1, \ldots, m$ place the drawing of $T_i$ to the right of the drawing of $T_{i-1}$ and at horizontal distance at least 1 from it.

Position the root half-way between the roots of $T_1$ and $T_m$. 

```
Level-based Layout - General trees

Algorithm Outline:
Input: A rooted tree
Output: A level-based drawing of $T$

Base case: A single vertex
Divide: Assume that $T$ has subtrees $T_1, \ldots, T_m$. Draw each $T_i$ recursively.
Conquer: For $i = 1, \ldots, m$ place the drawing of $T_i$ to the right of the drawing of $T_{i-1}$ and at horizontal distance at least 1 from it.
Position the root half-way between the roots of $T_1$ and $T_m$.
```

\[ T_1 \quad T_2 \quad T_m \]
**Algorithm Outline:**

**Input:** A rooted tree  
**Output:** A level-based drawing of \( T \)

**Base case:** A single vertex

**Divide:** Assume that \( T \) has subtrees \( T_1, \ldots, T_m \). Draw each \( T_i \) recursively.

**Conquer:**
- For \( i = 1, \ldots, m \) place the drawing of \( T_i \) to the right of the drawing of \( T_{i-1} \) and at horizontal distance at least 1 from it.
- Position the root half-way between the roots of \( T_1 \) and \( T_m \).

Questions?
Cons cell diagram in LISP.

*Cons*(constructs) are memory objects which hold two values or pointers to values.

![Diagram of cons cells of the simple tree.](http://gajon.org/)

**Figure 3:** Diagram of cons cells of the simple tree.

**HV-layout (Horizontal-Vertical)**
HV-Layout

Divide & Conquer Approach:
HV-Layout

Idea for binary trees:
- Children are vertically and horizontally aligned with the root
- The bounding boxes of the children do not intersect

Induction base:  •

Induction step: combine layouts

horizontal combination (Area: $3 \times 7$)

vertical combination (Area: $6 \times 4$)
HV-Layout

Idea for binary trees:
- Children are vertically and horizontally aligned with the root
- The bounding boxes of the children do not intersect

Induction base:  

Induction step: combine layouts

horizontal combination  
(Area: $3 \times 7$)

vertical combination  
(Area: $6 \times 4$)

Compute minimum area using Dynamic Programming
Right-Heavy HV-Layout

**Right-Heavy approach:**
- At every induction step apply horizontal combination
- Place the larger subtree to the right
Right-Heavy HV-Layout

**Right-Heavy approach:**
- At every induction step apply horizontal combination
- Place the larger subtree to the right

**Lemma**

Let $T$ be a binary tree. The height of the drawing constructed by Right-Heavy approach is at most $\log n$. 
Right-Heavy HV-Layout

Right-Heavy approach:
- At every induction step apply horizontal combination
- Place the larger subtree to the right

Lemma
Let $T$ be a binary tree. The height of the drawing constructed by Right-Heavy approach is at most $\log n$.

Proof:
- Each vertical edge has length 1
- Let $w$ be the lowest node in the drawing
- Let $P$ be a path from $w$ to the root of $T$
- For every edge $(u, v)$ in $P$: $|T(v)| > 2|T(u)|$
- $\Rightarrow P$ contains at most $\log n$ edges
Theorem

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:
Theorem

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:

- $\Gamma$ is HV-drawing (planar, orthogonal)
Right-Heavy HV-Layout

Theorem

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:

- $\Gamma$ is HV-drawing (planar, orthogonal)
- The width of $\Gamma$ is at most
Theorem

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:

- $\Gamma$ is HV-drawing (planar, orthogonal)
- The width of $\Gamma$ is at most $n-1$
Right-Heavy HV-Layout

**Theorem**

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:

- $\Gamma$ is HV-drawing (planar, orthogonal)
- The width of $\Gamma$ is at most $n-1$
- The height is at most $\log n$
Right-Heavy HV-Layout

**Theorem**

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:

- $\Gamma$ is HV-drawing (planar, orthogonal)
- The width of $\Gamma$ is at most $n-1$
- The height is at most $\log n$
- The area is $O(n \log n)$
Right-Heavy HV-Layout

Theorem

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:

- $\Gamma$ is HV-drawing (planar, orthogonal)
- The width of $\Gamma$ is at most $n-1$
- The height is at most $\log n$
- The area is $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings, up to translation
Right-Heavy HV-Layout

**Theorem**

Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:

- $\Gamma$ is HV-drawing (planar, orthogonal)
- The width of $\Gamma$ is at most $n-1$
- The height is at most $\log n$
- The area is $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings, up to translation

**General rooted tree:**

![Diagram of a general rooted tree with a highlighted largest subtree]
Right-Heavy HV-Layout

Theorem
Let $T$ be a binary tree with $n$ vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ such that:
- $\Gamma$ is HV-drawing (planar, orthogonal)
- The width of $\Gamma$ is at most $n-1$
- The height is at most $\log n$
- The area is $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings, up to translation

General rooted tree:

Questions?
Applications

Radial layout

An unrooted phylogenetic tree for myosin, a superfamily of proteins.

"A myosin family tree“ Journal of Cell Science
Radial Layout
Radial Layout
Radial Layout

**Example:** Angle corresponding to the subtree rooted at $u$: $\tau_u = \frac{\ell(u)}{\ell(v) - 1}$
Radial Layout

**Example:**

Angle corresponding to the subtree rooted at $u$: $\tau_u = \frac{\ell(u)}{\ell(v)-1}$

![Diagram](image-url)
Radial Layout

**Example:**  
Angle corresponding to the subtree rooted at $u$:  
$$\tau_u = \frac{\ell(u)}{\ell(v) - 1}$$
Radial Layout

**Example:**

- Angle corresponding to the subtree rooted at $u$:
  
  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$

![Diagram of a radial layout with labels and angles](image)
Example: Angle corresponding to the subtree rooted at $u$: $	au_u = \frac{\ell(u)}{\ell(v) - 1}$

Radial Layout
Radial Layout

**Example:**

Angle corresponding to the subtree rooted at $u$: $\tau_u = \frac{\ell(u)}{\ell(v) - 1}$
Radial Layout

**Example:**

Angle corresponding to the subtree rooted at $u$: $\tau_u = \frac{\ell(u)}{\ell(v) - 1}$

\[
\begin{align*}
\ell(u) & \quad v \\
u & \quad \bullet
\end{align*}
\]
Radial Layout

Example: Angle corresponding to the subtree rooted at $u$: $\tau_u = \frac{\ell(u)}{\ell(v) - 1}$
Radial Layout

How to avoid crossings:
Radial Layout

How to avoid crossings:
Radial Layout

How to avoid crossings:
Radial Layout

How to avoid crossings:
Radial Layout

How to avoid crossings:
Radial Layout

How to avoid crossings:

- $\tau_u$ - angle of the wedge corresponding to vertex $u$
- $\rho_i$ - radius of layer $i$
- $\ell(v)$ - number of nodes in the subtree rooted at $v$
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
How to avoid crossings:

- $\tau_u$ - angle of the wedge corresponding to vertex $u$
- $\rho_i$ - radius of layer $i$
- $\ell(v)$ - number of nodes in the subtree rooted at $v$
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
Radial Layout

How to avoid crossings:

- $\tau_u$ - angle of the wedge corresponding to vertex $u$
- $\rho_i$ - radius of layer $i$
- $\ell(v)$-number of nodes in the subtree rooted at $v$
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$

$$\tau_u = \min\left\{ \frac{\ell(u)}{\ell(v)-1}, 2 \arccos \frac{\rho_i}{\rho_{i+1}} \right\}$$

(correction)
Radial Layout

How to avoid crossings:

- $\tau_u$ - angle of the wedge corresponding to vertex $u$
- $\rho_i$ - radius of layer $i$
- $\ell(v)$ - number of nodes in the subtree rooted at $v$
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_i + 1}$

$\tau_u = \min\left\{ \frac{\ell(u)}{\ell(v) - 1}, 2 \arccos \frac{\rho_i}{\rho_i + 1} \right\}$ (correction)

Alternatively use number of leaves in the subtree to subdivide the angles [book Di Battista et al.]
Radial Layout

How to avoid crossings:

- \( \tau_u \) - angle of the wedge corresponding to vertex \( u \)
- \( \rho_i \) - radius of layer \( i \)
- \( \ell(v) \) - number of nodes in the subtree rooted at \( v \)
- \( \cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}} \)

\[ \tau_u = \min\left\{ \frac{\ell(u)}{\ell(v)-1}, 2 \arccos \frac{\rho_i}{\rho_{i+1}} \right\} \] (correction)

Alternatively use number of leaves in the subtree to subdivide the angles [book Di Battista et al.]

Questions?
Other Visualization Styles

Writing Without Words: the project explores methods of visually-representing text and visualises the differences in writing styles when comparing different authors.
Other Visualization Styles

Writing Without Words: the project explores methods of visually-representing text and visualises the differences in writing styles when comparing different authors.

similar to Ballon layout
Other Visualization Styles

A phylogenetically organised display of data for all placental mammal species.

Fractal tree layout
for more applications and layouts...