Visualisation of graphs

Drawing trees and series-parallel graphs

Divide and conquer methods

Jonathan Klawitter · Summer semester 2020
Trees

- Tree - connected graph without cycles
- here: binary and rooted

![Tree diagram]

- $T(v)$
- $T_l(v)$
- $T_r(v)$
Trees

- Tree - connected graph without cycles
- here: binary and rooted

Tree traversal
Trees

- Tree - connected graph without cycles
- here: binary and rooted

**Tree traversal**
- Depth-first search

\[ T(v) = T_l(v) \cup \{v\} \cup T_r(v) \]
Trees

- Tree - connected graph without cycles
- here: binary and rooted

Tree traversal
- Depth-first search
  - Pre-order – first parent, then subtrees
Trees

- Tree - connected graph without cycles
- Here: binary and rooted

Tree traversal
- Depth-first search
  - Pre-order – first parent, then subtrees
  - In-order – left child, parent, right child
Trees

- Tree - connected graph without cycles
- Here: binary and rooted

Tree traversal
- Depth-first search
  - Pre-order – first parent, then subtrees
  - In-order – left child, parent, right child
  - Post-order – first subtrees, then parent
Trees

- Tree - connected graph without cycles
- Here: binary and rooted

**Tree traversal**

- 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
Trees

- Tree - connected graph without cycles
- Here: binary and rooted

**Tree traversal**
- 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

**Isomporphism**
- Simple
- Axial
Level-based layout – applications

Decision tree for outcome prediction after traumatic brain injury

Source: Nature Reviews Neurology
Level-based layout – applications

Family tree of LOTR elves and half-elves

Aloisius Gaultier 1821
Level-based layout – drawing style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimise?
Level-based layout – drawing style

Drawing conventions

- Vertices lie on layers and have integer coordinates
- Parent centred above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings

What are properties of the layout?
What are the drawing conventions?
What are aesthetics to optimise?
Level-based layout – drawing style

**What are properties of the layout?**
- Vertices lie on layers and have integer coordinates
- Parent centred above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings

**What are the drawing conventions?**
- Vertices lie on layers and have integer coordinates
- Parent centred above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings

**What are aesthetics to optimise?**
- Area
Level-based layout – algorithm

**Input:** A binary tree $T$

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

**Base case:**

**Divide:**

**Conquer:**
Level-based layout – algorithm

**Input:** A binary tree $T$

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

**Base case:** A single vertex ●

**Divide:**

**Conquer:**
Level-based layout – algorithm

**Input:** A binary tree $T$

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

**Base case:** A single vertex ●

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

**Conquer:**
Level-based layout – algorithm

**Input:** A binary tree $T$

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

**Base case:** A single vertex ●

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

**Conquer:**
Level-based layout – algorithm

**Input:** A binary tree $T$

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

**Base case:** A single vertex ●

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

**Conquer:**
Level-based layout – algorithm

**Input:** A binary tree $T$

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

**Base case:** A single vertex

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

**Conquer:**

![Diagram of a binary tree with a leveled drawing]
Level-based layout – algorithm

**Input:** A binary tree \( T \)

**Output:** A leveled drawing of \( T \)

**Base case:** A single vertex ●

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

**Conquer:**

- parent centered wrt to children
- some agreed distance
Level-based layout – algorithm

**Input:** A binary tree \( T \)

**Output:** A leveled drawing of \( T \)

**Base case:** A single vertex ●

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

**Conquer:**

- some agreed distance
- parent centered wrt to children
- sometimes 3 apart for grid drawing!
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child

**Phase 2 – preorder traversal:**
- Compute x- and y-coordinates
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child

**Phase 2 – preorder traversal:**
- Compute x- and y-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
  - Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute \( x \)- and \( y \)-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets

**Phase 2 – preorder traversal:**
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
  - $x$-offset($v_l$) = $-\lceil \frac{d_v}{2} \rceil$, $x$-offset($v_r$) = $\lceil \frac{d_v}{2} \rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
  - $x$-offset($v_l) = -\lceil \frac{d_v}{2} \rceil$, $x$-offset($v_r) = \lceil \frac{d_v}{2} \rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
  - \( x\text{-offset}(v_l) = -\left\lfloor \frac{d_v}{2} \right\rfloor \), \( x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
  - Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

**Phase 2 – preorder traversal:**
- Compute \( x \)- and \( y \)-coordinates

Runtime?
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
  - \( x\text{-offset}(v_l) = -\left\lceil \frac{d_v}{2} \right\rceil, \ x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
  - Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

Phase 2 – preorder traversal:
- Compute \( x \)- and \( y \)-coordinates

Runtime?
- How often do we have to walk along a contour?
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
  - \( x\text{-offset}(v_l) = -\left\lceil \frac{d_v}{2} \right\rceil \), \( x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
  - Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

**Phase 2 – preorder traversal:**
- Compute x- and y-coordinates

**Runtime?**
- How often do we have to walk along a contour?
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
  - \( x_{-\text{offset}}(v_l) = -\left\lceil \frac{d_v}{2} \right\rceil, \quad x_{-\text{offset}}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
  - Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

**Phase 2 – preorder traversal:**
- Compute \( x \)- and \( y \)-coordinates

**Runtime?**
- How often do we have to walk along a contour?
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
  - $x$-offset($v_l$) = $-\lceil \frac{d_v}{2} \rceil$, $x$-offset($v_r$) = $\lceil \frac{d_v}{2} \rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

**Phase 2 – preorder traversal:**
- Compute $x$- and $y$-coordinates

**Runtime?**
- How often do we have to walk along a contour?
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
  - $x\text{-offset}(v_l) = -\left\lfloor \frac{d_v}{2} \right\rfloor$, $x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \min.$ horiz. distance between $v_l$ and $v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates

Runtime?
- How often do we have to walk along a contour?
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
  - $x\text{-offset}(v_l) = -\left\lfloor \frac{d_v}{2} \right\rfloor$, $x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates

Runtime?
- How often do we have to walk along a contour?
Level-based layout – algorithm details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
  - $x$-offset($v_l) = -\left\lfloor \frac{d_v}{2} \right\rfloor$, $x$-offset($v_r) = \left\lceil \frac{d_v}{2} \right\rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
  - Contour is linked list of vertex coordinates/offsets
- Find $d_v = \min$ horiz. distance between $v_l$ and $v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates

Runtime?
- How often do we have to walk along a contour?
Level-based layout – algorithm details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
  - \( x\text{-offset}(v_l) = -\left\lceil \frac{d_v}{2} \right\rceil, \ x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
  - Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

**Phase 2 – preorder traversal:**
- Compute \( x \)- and \( y \)-coordinates

**Runtime?**
- How often do we have to walk along a contour?
  \( \Rightarrow O(n) \)
Theorem. (Reingold & Tilford ’81)

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is leveled: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Vertical and horizontal distances are at least 1
- Each vertex is centred wrt its children
- Area of $\Gamma$ is in $O(n^2)$
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic trees have congruent drawings, up to translation and reflection around y-axis
Level-based layout – result

**Theorem.** (Reingold & Tilford ’81)
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is leveled: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Vertical and horizontal distances are at least 1
- Each vertex is centred wrt its children
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic trees have congruent drawings, up to translation and reflection around y-axis
Level-based layout – result

**Theorem.** (Reingold & Tilford ’81)
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:
- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is leveled: $y$-coordinate of vertex $v$ is $-\text{depth}(v)$
- Vertical and horizontal distances are at least 1
- Each vertex is centred wrt its children
- Area of $\Gamma$ is in $O(n^2)$
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic trees have congruent drawings, up to translation and reflection around y-axis
Level-based layout – result

**Theorem.** (Reingold & Tilford ’81)
Let \( T \) be a binary tree with \( n \) vertices. We can construct a drawing \( \Gamma \) of \( T \) in \( \mathcal{O}(n) \) time, such that:
- \( \Gamma \) is planar, straight-line and strictly downward
- \( \Gamma \) is leveled: y-coordinate of vertex \( v \) is \(-\text{depth}(v)\)
- Vertical and horizontal distances are at least 1
- Each vertex is centred wrt its children
- Area of \( \Gamma \) is in \( \mathcal{O}(n^2) \)

Example?
Theorem. (Reingold & Tilford ’81)

Let \( T \) be a binary tree with \( n \) vertices. We can construct a drawing \( \Gamma \) of \( T \) in \( O(n) \) time, such that:

- \( \Gamma \) is planar, straight-line and strictly downward
- \( \Gamma \) is leveled: y-coordinate of vertex \( v \) is \(-\)depth\( (v)\)
- Vertical and horizontal distances are at least 1
- Each vertex is centred wrt its children
- Area of \( \Gamma \) is in \( O(n^2) \)
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic trees have congruent drawings, up to translation and reflection around y-axis
Theorem. (Reingold & Tilford '81)
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:
- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is leveled: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Vertical and horizontal distances are at least 1
- Each vertex is centred wrt its children
- Area of $\Gamma$ is in $O(n^2)$
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic trees have congruent drawings, up to translation and reflection around y-axis
Level-based layout – area

- Presented algorithm tries to minimise width
Level-based layout – area

- Presented algorithm tries to minimise width
- Does not always achieve that!
Level-based layout – area

- Presented algorithm tries to minimise width
- Does not always achieve that!
- Divide-and-conquer strategy cannot achieve optimal width
Level-based layout – area

- Presented algorithm tries to minimise width
- Does not always achieve that!
- Divide-and-conquer strategy cannot achieve optimal width

- Drawing with min width (but without the grid) can be constructed by an LP
Level-based layout – area

- Presented algorithm tries to minimise width
- Does not always achieve that!
- Divide-and-conquer strategy cannot achieve optimal width

Drawing with min width (but without the grid) can be constructed by an LP
Problem is NP-hard on grid
Applications

- Cons cell diagram in LIPS
- Cons(constructs) are memory objects which hold two values or pointers to values

Source: after gajon.org/trees-linked-lists-common-lisp/
Drawing-style: hv-drawings

Applications
- Cons cell diagram in LIPS
- Cons(constructs) are memory objects which hold two values or pointers to values

Drawing conventions
- Children are vertically and horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint

Drawing aesthetics
- Height, width, area

Source: after gajon.org/trees-linked-lists-common-lisp/
Drawing-style: hv-drawings

Applications
- Cons cell diagram in LIPS
- Cons(constructs) are memory objects which hold two values or pointers to values

```
1  3  /  5  10  11 /
   |  1 /  9  12 /
4  6  7  8 /
```

Drawing conventions
- Children are vertically and horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint

Drawing aesthetics

Source: after gajon.org/trees-linked-lists-common-lisp/
Drawing-style: hv-drawings

Applications
- Cons cell diagram in LIPS
- Cons(constructs) are memory objects which hold two values or pointers to values

Drawing conventions
- Children are vertically and horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint

Drawing aesthetics
- Height, width, area

Source: after gajon.org/trees-linked-lists-common-lisp/
hv-drawings – algorithm

Input: A binary tree $T$
Output: A hv-drawing of $T$

Base case:  ●
Divide: Recursively apply the algorithm to draw the left and right subtrees

Conquer:
hv-drawings – algorithm

**Input:** A binary tree $T$

**Output:** A hv-drawing of $T$

**Base case:**

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

**Conquer:**

- horizontal combination
- vertical combination
hv-drawing – right-heavy hv-layout

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
hv-drawing – right-heavy hv-layout

**Right-heavy approach**
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
hv-drawing – right-heavy hv-layout

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
hv-drawing – right-heavy hv-layout

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
hv-drawing – right-heavy hv-layout

**Right-heavy approach**
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
hv-drawing – right-heavy hv-layout

**Right-heavy approach**
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
hv-drawing – right-heavy hv-layout

Right-heavy approach

■ Always apply horizontal combination
■ Place the larger subtree to the right
  ■ Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$. 

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree $:= \text{number of vertices}$
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
■ width at most $n - 1$ and
■ height at most $\log n$.

Right-heavy approach
■ Always apply horizontal combination
■ Place the larger subtree to the right
■ Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$. 

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right
- Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

**Right-heavy approach**
- Always apply horizontal combination
- Place the larger subtree to the right
- Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices

---

Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

---

at least \cdot 2

at least \cdot 2
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$. 

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

**Right-heavy approach**
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$.
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  - Size of subtree := number of vertices

How to implement this in linear time?
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy
algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$
s.t.:
- $\Gamma$ is hv-drawing (planar, orthogonal)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is hv-drawing (planar, orthogonal)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:
- $\Gamma$ is hv-drawing (planar, orthogonal)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is hv-drawing (planar, orthogonal)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation

General rooted tree

Optimal area?
- Not with divide & conquer approach, but
- can be computed with Dynamic Programming.
Radial layout – applications

Phylogenetic tree
by Colicelli, ScienceSignaling, 2004
Radial layout – applications

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

Greek Myth Family by Ribbecca, 2011
Radial layout – drawing style

**Drawing conventions**
- Vertices lie on circular layers according to their depth
- Drawing is planar

**Drawing aesthetics**
- Distribution of the vertices
Radial layout – drawing style

**Drawing conventions**
- Vertices lie on circular layers according to their depth
- Drawing is planar

**Drawing aesthetics**
- Distribution of the vertices

How may an algorithm optimise the distribution of the vertices?
Radial layout – algorithm attempt

Idea

- Angle corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \frac{\ell(u)}{\ell(v) - 1}$$
Radial layout – algorithm attempt

Idea

- Angle corresponding to size $\ell(u)$ of $T(u)$:

$$ \tau_u = \frac{\ell(u)}{\ell(v) - 1} $$
Radial layout – algorithm attempt

Idea

- Angle corresponding to size $\ell(u)$ of $T(u)$:

$$
\tau_u = \frac{\ell(u)}{\ell(v) - 1}
$$
Radial layout – algorithm attempt

Idea
- Angle corresponding to size \( \ell(u) \) of \( T(u) \):

\[
\tau_u = \frac{\ell(u)}{\ell(v) - 1}
\]
Radial layout – algorithm attempt

Idea

- Angle corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \frac{\ell(u)}{\ell(v) - 1}$$
Radial layout – algorithm attempt

**Idea**

- Angle corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \frac{\ell(u)}{\ell(v) - 1}$$
Radial layout – algorithm attempt

**Idea**

- Angle corresponding to size $\ell(u)$ of $T(u)$:

\[
\tau_u = \frac{\ell(u)}{\ell(v) - 1}
\]
Radial layout – algorithm attempt

Idea

- Angle corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \frac{\ell(u)}{\ell(v) - 1}$$
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

\[ \tau_u \] – angle of the wedge corresponding to vertex \( u \)
Radial layout – how to avoid crossings

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

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
- $\rho_i$ – radius of layer $i$

\[
\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}
\]

\[
\tau_u = \min\left\{ \frac{\ell(u)}{\ell(v)} - 1, 2 \arccos \frac{\rho_i}{\rho_{i+1}} \right\}
\]
Radial layout – how to avoid crossings

- \( \tau_u \) – angle of the wedge corresponding to vertex \( u \)
- \( \ell(u) \) – number of nodes in the subtree rooted at \( u \)
- \( \rho_i \) – radius of layer \( i \)

\[
\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}
\]

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

Alternative:

\[
\alpha_{\min} = \alpha_v - \arccos \frac{\rho_i}{\rho_{i+1}}
\]

\[
\alpha_{\max} = \alpha_v + \arccos \frac{\rho_i}{\rho_{i+1}}
\]
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)

begin
    postorder($r$)
    preorder($r$, 0, 0, $2\pi$)
    return $(d_v, \alpha_v)_{v \in V(T)}$
    // vertex pos./polar coord.
end

postorder(vertex $v$)

    calculate the size of the subtree recursively
Radial layout – pseudocode

RadialTreeLayout(\textit{tree} \(T\), \textit{root} \(r \in T\), radii \(\rho_1 < \cdots < \rho_k\))

\begin{algorithm}
\textbf{begin}
  \begin{algorithmic}
    \State \textit{postorder}\((r)\)
    \State \textit{preorder}\((r, 0, 0, 2\pi)\)
    \State \textbf{return} \((d_v, \alpha_v)_{v \in V(T)}\)
    \State // vertex pos./polar coord.
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\textbf{postorder\(\textit{vertex} \ v\)}
\begin{algorithmic}
  \State \(\ell(v) \leftarrow 1\)
  \State \textbf{foreach} child \(w\) of \(v\) \textbf{do}
    \State \textit{postorder}\((w)\)
    \State \(\ell(v) \leftarrow \ell(v) + \ell(w)\)
\end{algorithmic}
\end{algorithm}
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)

begin
postorder($r$)
preorder($r, 0, 0, 2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$

// vertex pos./polar coord.

postorder(vertex $v$)

\[ \ell(v) \leftarrow 1 \]

foreach child $w$ of $v$

do
postorder($w$)

\[ \ell(v) \leftarrow \ell(v) + \ell(w) \]

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)

\[ d_v \leftarrow \rho_t \]

\[ \alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}}) / 2 \]

if $t > 0$ then

\[ \alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\} \]

\[ \alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\} \]

left $\leftarrow \alpha_{\text{min}}$

foreach child $w$ of $v$

do
right $\leftarrow$ left $+ \frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$

preorder($w, t+1, \text{left, right}$)

left $\leftarrow$ right
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)

\begin{verbatim}
begin
  postorder($r$)
  preorder($r$, 0, 0, $2\pi$)
  return $(d_v, \alpha_v)_{v \in V(T)}$
  // vertex pos./polar coord.

postorder(vertex $v$)

  $\ell(v) \leftarrow 1$

  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$
end
\end{verbatim}

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)

\begin{verbatim}
$\ell(v) \leftarrow 1$
$\ell(v) \leftarrow \ell(v) + \ell(w)$

if $t > 0$ then
  $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
  $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$

left $\leftarrow \alpha_{\text{min}}$

foreach child $w$ of $v$ do
  right $\leftarrow left + \frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
  preorder($w$, $t + 1$, left, right)
left $\leftarrow$ right
\end{verbatim}
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)

begin
  
  postorder($r$)
  preorder($r, 0, 0, 2\pi$)
  return $(d_v, \alpha_v)_{v \in V(T)}$

  // vertex pos./polar coord.

postorder(vertex $v$
  
  $\ell(v) \leftarrow 1$
  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$

end

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$

  $d_v \leftarrow \rho_t$
  $\alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2$
  //output

  if $t > 0$ then
    $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
    $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$

    left $\leftarrow \alpha_{\text{min}}$
    foreach child $w$ of $v$ do
      right $\leftarrow left + \frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
      preorder($w, t+1, left, right$)
    left $\leftarrow right$
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)

begin

postorder($r$)

preorder($r, 0, 0, 2\pi$)

return ($d_v, \alpha_v$) $\forall v \in V(T)$ // vertex pos./polar coord.

postorder(vertex $v$)

$\ell(v) \leftarrow 1$

foreach child $w$ of $v$ do

postorder($w$)

$\ell(v) \leftarrow \ell(v) + \ell(w)$

end

preorder(vertex $v$, $t$, $\alpha_{\min}$, $\alpha_{\max}$)

$d_v \leftarrow \rho_t$

$\alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max}) / 2$ //output

if $t > 0$ then

$\alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$

$\alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$

left $\leftarrow \alpha_{\min}$

foreach child $w$ of $v$ do

right $\leftarrow$ left $+ \frac{\ell(w)}{\ell(v) - 1} \cdot (\alpha_{\max} - \alpha_{\min})$

preorder($w$, $t + 1$, left, right)

left $\leftarrow$ right

Runtime?
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  postorder($r$)
  preorder($r$, 0, 0, $2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex pos./polar coord.
postorder(vertex $v$)
  $\ell(v) \leftarrow 1$
  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$
preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)
\begin{align*}
  d_v &\leftarrow \rho_t \\
  \alpha_v &\leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2 \quad // output \\
  \text{if } t > 0 \text{ then} \\
  \quad \alpha_{\text{min}} &\leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\} \\
  \quad \alpha_{\text{max}} &\leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\} \\
  \quad \text{left } &\leftarrow \alpha_{\text{min}} \\
  \quad \text{foreach child } w \text{ of } v \text{ do} \\
  \quad \quad \text{right } &\leftarrow \text{left} + \frac{\ell(w)}{\ell(v) - 1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}}) \\
  \quad \quad \text{preorder}(w, t + 1, \text{left}, \text{right}) \\
  \quad \quad \text{left } &\leftarrow \text{right}
\end{align*}

Runtime? $\mathcal{O}(n)$
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)

\[
\begin{align*}
\text{begin} & \quad \text{postorder}(r) \\
& \quad \text{preorder}(r, 0, 0, 2\pi) \\
& \quad \text{return} \ (d_v, \alpha_v)_{v \in V(T)} \\
& \quad \text{// vertex pos./polar coord.}
\end{align*}
\]

postorder(vertex $v$)

\[
\ell(v) \leftarrow 1 \\
\text{foreach child $w$ of $v$ do} \\
& \quad \text{postorder}(w) \\
& \quad \ell(v) \leftarrow \ell(v) + \ell(w)
\]

preorder(vertex $v$, $t$, $\alpha_{\min}$, $\alpha_{\max}$)

\[
\begin{align*}
& d_v \leftarrow \rho_t \\
& \alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2 \\
& \text{//output}
\end{align*}
\]

\[
\begin{align*}
& \text{if } t > 0 \text{ then} \\
& \quad \alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\} \\
& \quad \alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\} \\
& \quad \text{left} \leftarrow \alpha_{\min} \\
& \quad \text{foreach child $w$ of $v$ do} \\
& \quad & \quad \text{right} \leftarrow \text{left} + \frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\max} - \alpha_{\min}) \\
& \quad & \quad \text{preorder}(w, t + 1, \text{left}, \text{right}) \\
& \quad & \quad \text{left} \leftarrow \text{right}
\end{align*}
\]

Runtime? $O(n)$

Correctness?
Radial layout – pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
postorder($r$)
preorder($r$, 0, 0, $2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex pos./polar coord.
dv ← $\rho_t$
$\alpha_v ← (\alpha_{\min} + \alpha_{\max}) / 2$
//output
if $t > 0$ then
$\alpha_{\min} ← \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
$\alpha_{\max} ← \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
left ← $\alpha_{\min}$
foreach child $w$ of $v$ do
postorder($w$)
right ← left + $\frac{\ell(w)}{\ell(v)} \cdot (\alpha_{\max} - \alpha_{\min})$
preorder($w$, $t + 1$, left, right)
left ← right
postorder(vertex $v$)
$\ell(v) ← 1$
foreach child $w$ of $v$ do
postorder($w$)
$\ell(v) ← \ell(v) + \ell(w)$
Runtime? $\mathcal{O}(n)$
Correctness? ✓
Theorem.
Let $T$ be a tree with $n$ vertices. The RadialTreeLayout algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:
- $\Gamma$ is radial drawing
- Vertices lie on circle according to their depth
- Area quadratic in max degree times height of $T$
(see book if interested)
Other tree visualisation styles

Writing Without Words: The project explores methods to visualises the differences in writing styles of different authors.

Similar to ballon layout
Other tree visualisation styles

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

Fractal layout
Other tree visualisation styles
Other tree visualisation styles
A graph $G$ is **series-parallel**, if
- it contains a single edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1$, $G_2$ with sources $s_1$, $s_2$ and sinks $t_1$, $t_2$ that are combined using one of the following rules:

**Series composition**

$G_1$  
\[ s_1 \quad t_1 \]
\[ s_2 \quad t_2 \]

$G_2$  
\[ s_1 \quad t_2 \]

$G_1 G_2$  
\[ s_1 \quad t_1 = s_2 \]

**Parallel composition**

$G_1$  
\[ s_1 \quad t_1 = t_2 \]

$G_2$  
\[ s_2 \quad t_1 = s_2 \]
Series-parallel graphs

A graph $G$ is **series-parallel**, if

- it contains a single edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1$, $G_2$ with sources $s_1$, $s_2$ and sinks $t_1$, $t_2$ that are combined using one of the following rules:

**Series composition**

**Parallel composition**

convince yourself that series-parallel graphs are planar
Series-parallel graphs – decomposition tree

A decomposition tree of $G$ is a binary tree $T$ with nodes of three types: $S$, $P$ and $Q$-type.
A **decomposition tree** of $G$ is a binary tree $T$ with nodes of three types: **S**, **P** and **Q**-type

- A Q-node represents a single edge
A decomposition tree of $G$ is a binary tree $T$ with nodes of three types: $S$, $P$ and $Q$-type.

- A Q-node represents a single edge.
- An S-node represents a series composition; its children $T_1$ and $T_2$ represent $G_1$ and $G_2$. 

![Diagram of a decomposition tree with nodes labeled S, P, and Q, and subgraphs $G_1$ and $G_2$.]
**Series-parallel graphs – decomposition tree**

A decomposition tree of $G$ is a binary tree $T$ with nodes of three types: **S**, **P** and **Q**-type

- A Q-node represents a single edge
- An S-node represents a series composition; its children $T_1$ and $T_2$ represent $G_1$ and $G_2$
- A P-node represents a parallel composition; its children $T_1$ and $T_2$ represent $G_1$ and $G_2
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – decomposition example
Series-parallel graphs – applications

Flowcharts

PERT-Diagrams

(Program Evaluation and Review Technique)
Series-parallel graphs – applications

Flowcharts
PERT-Diagrams
(Program Evaluation and Review Technique)

Computational complexity:
Linear time algorithms for $NP$-hard problems
(e.g. Maximum Matching, MIS, Hamiltonian Completion)
Series-parallel graphs – drawing style

Drawing conventions

Drawing aesthetics
Series-parallel graphs – drawing style

**Drawing conventions**
- Planarity
- Straight-line edges
- Upward

**Drawing aesthetics**
Series-parallel graphs – drawing style

**Drawing conventions**
- Planarity
- Straight-line edges
- Upward

**Drawing aesthetics**
- Area
- Symmetry
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$.
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree
- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition

![Diagram of series-parallel graphs](image)
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree
- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes  

Divide: Draw $G_1$ and $G_2$ first

Conquer:
- S-nodes / series composition
- P-nodes / parallel composition
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes  
Divide: Draw $G_1$ and $G_2$ first

Conquer:
- S-nodes / series composition
- P-nodes / parallel composition

Divide: Draw $G_1$ and $G_2$ first

Do you see any problem?
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition

Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition

Single edge
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree
- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:
- S-nodes / series composition
- P-nodes / parallel composition

change embedding!
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition

Divide: Draw $G_1$ and $G_2$ first

Change embedding!
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes  Divide: Draw $G_1$ and $G_2$ first

Conquer:
- S-nodes / series composition
- P-nodes / parallel composition

Divide:

\[
\Delta(G_1) \\
\Delta(G_2)
\]

Change embedding!
Series-parallel graphs – straight-line drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

**Base case:** Q-nodes  
**Divide:** Draw $G_1$ and $G_2$ first

**Conquer:**
- S-nodes / series composition
- P-nodes / parallel composition

Divide:
- Draw $G_1$ and $G_2$ first

Conquer:
- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

change embedding!
Series-parallel graphs – straight-line drawings

What makes parallel composition possible without creating crossings?
Series-parallel graphs – straight-line drawings

What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
Series-parallel graphs – straight-line drawings

What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
Series-parallel graphs – straight-line drawings

What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
Series-parallel graphs – straight-line drawings

What makes parallel composition possible without creating crossings?

Assume the following holds:
the only vertex in $\angle(v)$ is $s$

Assume $\pi/4$
Series-parallel graphs – straight-line drawings

■ What makes parallel composition possible without creating crossings?

![Diagram showing parallel composition without crossings]

Assume the following holds:
the only vertex in angle(\(v\)) is \(s\)

■ This condition is preserved during the induction step.
Series-parallel graphs – straight-line drawings

■ What makes parallel composition possible without creating crossings?

Assume the following holds:
the only vertex in angle(v) is s

■ This condition is preserved during the induction step.

Lemma.
The drawing produced by the algorithm is planar.
Series-parallel graphs – result

Theorem.
Let $G$ be a series-parallel graph. Then $G$ (with variable embedding) admits a drawing $\Gamma$ that
- is upward planar and
- a straight-line drawing
- with area in $O(n^2)$.
- Isomorphic components of $G$ have congruent drawings up to translation.
$\Gamma$ can be computed in $O(n)$ time.
Series-parallel graphs – fixed embedding

**Theorem.** [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Series-parallel graphs – fixed embedding

**Theorem.** [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
**Theorem.** [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Series-parallel graphs – fixed embedding

**Theorem.** [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Series-parallel graphs – fixed embedding

**Theorem.** [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94] There exists a $2^n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

\[ 2 \cdot \text{Area}(G_n) < \text{Area}(\Pi) \]
Theorem. [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

$$2 \cdot \text{Area}(G_n) < \text{Area}(\Pi)$$
Series-parallel graphs – fixed embedding

**Theorem.** [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

$2 \cdot \text{Area}(G_n) < \text{Area}(\Pi)$
Series-parallel graphs – fixed embedding

**Theorem.** [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

- $2 \cdot \text{Area}(G_n) < \text{Area}(\Pi)$
- $2 \cdot \text{Area}(\Pi) \leq \text{Area}(G_{n+1})$
Series-parallel graphs – fixed embedding

Theorem. [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

- $2 \cdot \text{Area}(G_n) < \text{Area}(\Pi)$
- $2 \cdot \text{Area}(\Pi) \leq \text{Area}(G_{n+1})$
- $4 \cdot \text{Area}(G_n) \leq \text{Area}(G_{n+1})$
Literature

- [GD Ch. 3.1] for divide and conquer methods for rooted trees
- [RT81] Reingold and Tilford, "Tidier Drawings of Trees" 1981 – original paper for level-based layout algo
- treewis.net – compendium of drawing methods for trees (links on website)
- [GD Ch. 3.2] for divide and conquer methods for series-parallel graphs