Algorithms for graph visualization

Layouts for planar graphs. Shift method.

WINTER SEMESTER 2012/2013

Tamara Mchedlidze – Martin Nöllenburg – Ignaz Rutter
Straight line drawing of a planar graph
History

- Straight line drawing of a planar graph
History

- Straight line drawing of a planar graph

Theorem [Wagner ‘36, Fary ‘48, Stein ‘51]

Every planar graph has a planar straight-line drawing.
History

- Straight line drawing of a planar graph

Theorem [Wagner ’36, Fary ’48, Stein ’51]

Every planar graph has a planar straight-line drawing.

- These algorithms produce drawings with area not bounded by any polynomial on $n$. 
This lecture:

Theorem [De Fraysseix, Pach, Pollack ’90]

Every $n$-vertex planar graph has a planar straight-line drawing of a size $(2n - 4) \times (n - 2)$.

Next lecture:

Theorem [Schnyder ’90]

Every $n$-vertex planar graph has a planar straight-line drawing of a size $(n - 2) \times (n - 2)$. 
Canonical Ordering

Definition: Canonical Ordering

Let $G = (V, E)$ be a triangulated planar embedded graph of $n \geq 3$ vertices. An ordering $\pi = (v_1, v_2, \ldots, v_n)$ is called a canonical ordering, if the following conditions hold for each $k$, $3 \leq k \leq n$.

- (C1) Vertices $\{v_1, \ldots v_k\}$ induce a 2-connected internally triangulated graph, call it $G_k$. 
Canonical Ordering

Definition: Canonical Ordering

Let $G = (V, E)$ be a triangulated planar embedded graph of $n \geq 3$ vertices. An ordering $\pi = (v_1, v_2, \ldots, v_n)$ is called a canonical ordering, if the following conditions hold for each $k$, $3 \leq k \leq n$.

- (C1) Vertices $\{v_1, \ldots, v_k\}$ induce a 2-connected internally triangulated graph, call it $G_k$

- (C2) Edge $(v_1, v_2)$ belongs to the outer face of $G_k$
Definition: Canonical Ordering

Let \( G = (V, E) \) be a triangulated planar embedded graph of \( n \geq 3 \) vertices. An ordering \( \pi = (v_1, v_2, \ldots, v_n) \) is called a canonical ordering, if the following conditions hold for each \( k, 3 \leq k \leq n \).

- (C1) Vertices \( \{v_1, \ldots, v_k\} \) induce a 2-connected internally triangulated graph, call it \( G_k \).
- (C2) Edge \( (v_1, v_2) \) belongs to the outer face of \( G_k \).
- (C3) If \( k < n \) then vertex \( v_{k+1} \) lies in the outer face of \( G_k \), and all neighbors of \( v_{k+1} \) in \( G_k \) appear on the boundary of \( G_k \) consecutively.
Example of Canonical Ordering
Example of Canonical Ordering

$G_{16}$
Example of Canonical Ordering
Example of Canonical Ordering
Example of Canonical Ordering
Example of Canonical Ordering

$G_{14}$

14

15

16
Example of Canonical Ordering
Example of Canonical Ordering
Example of Canonical Ordering
Example of Canonical Ordering
Example of Canonical Ordering
Example of Canonical Ordering
Example of Canonical Ordering
Lemma

Every triangulated plane graph has a canonical ordering.

Let $G_n = G$, and let $v_1, v_2, v_n$ be the vertices of the outer face of $G_n$. Conditions C1-C3 hold.
Canonical Ordering Existence

Lemma

Every triangulated plane graph has a canonical ordering.

- Let $G_n = G$, and let $v_1, v_2, v_n$ be the vertices of the outer face of $G_n$. Conditions C1-C3 hold.
- Induction hypothesis: vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k + 1 \leq i \leq n$. 
Lemma

Every triangulated plane graph has a canonical ordering.

- Let $G_n = G$, and let $v_1, v_2, v_n$ be the vertices of the outer face of $G_n$. Conditions C1-C3 hold.
- Induction hypothesis: vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k + 1 \leq i \leq n$.
- Consider $G_k$. We search for $v_k$. 

![Diagram of a triangulated plane graph with a vertex $v_k$ highlighted]
Lemma

Every triangulated plane graph has a canonical ordering.

- Let $G_n = G$, and let $v_1, v_2, v_n$ be the vertices of the outer face of $G_n$. Conditions C1-C3 hold.
- Induction hypothesis: vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k + 1 \leq i \leq n$.
- Consider $G_k$. We search for $v_k$. 
Every triangulated plane graph has a canonical ordering.

Let $G_n = G$, and let $v_1, v_2, v_n$ be the vertices of the outer face of $G_n$. Conditions C1-C3 hold.

Induction hypothesis: vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k + 1 \leq i \leq n$.

Consider $G_k$. We search for $v_k$.

$v_k$ should not be adjacent to a chord
Lemma

Every triangulated plane graph has a canonical ordering.

- Let $G_n = G$, and let $v_1, v_2, v_n$ be the vertices of the outer face of $G_n$. Conditions C1-C3 hold.
- Induction hypothesis: vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k + 1 \leq i \leq n$.
- Consider $G_k$. We search for $v_k$.

$v_k$ should not be adjacent to a chord

Is it sufficient?
Canonical Ordering Existence

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.

- $v_k$

$$G_{k-1}$$
Canonical Ordering Existence

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.
Canonical Ordering Existence

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.

$$G_{k-1}$$

![Graph Diagram]

\[ v_k \]
**Canonical Ordering Existence**

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.
Canonical Ordering Existence

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.
Canonical Ordering Existence

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.

![Diagram showing the graph $G_{k-1}$ and removing $v_k$ results in a graph that is not biconnected.](image)
**Canonical Ordering Existence**

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.

- **Graph $G_{k-1}$:**
  - If $v_k$ is not adjacent to a chord, removal of $v_k$ leaves the graph biconnected.
  - **Diagram:**
    - $v_k$ is not connected to a chord.
    - Removal of $v_k$ results in a biconnected graph.

- **Graph $G'_{k-1}$:**
  - If $v_k$ is adjacent to a chord, removal of $v_k$ results in a non-biconnected graph.
  - **Diagram:**
    - $v_k$ is connected to a chord.
    - Removal of $v_k$ does not leave the graph biconnected.
Canonical Ordering Existence

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.
Canonical Ordering Existence

**Statement** If \( v_k \) is not adjacent to a chord then removal of \( v_k \) leaves the graph biconnected.

\[
\begin{align*}
G_{k-1} & \quad v_k \\
\text{not biconnected} & \\
G_{k-1} & \quad v_k \\
\text{not triangulated} & \\
\end{align*}
\]
Canonical Ordering Existence

**Statement** If $v_k$ is not adjacent to a chord then removal of $v_k$ leaves the graph biconnected.

- $G_{k-1}$ not biconnected
- $G_{k-1}$ not triangulated
- A chord!
Why a vertex not adjacent to a chord exists?
Why a vertex not adjacent to a chord exists?
Computing Canonical Ordering

Algorithm CO

forall the \( v \in V \) do
\[\text{chords}(v) \leftarrow 0; \text{out}(v) \leftarrow \text{false}; \text{mark}(v) \leftarrow \text{false};\]
\(\text{out}(v_1), \text{out}(v_2), \text{out}(v_n) \leftarrow \text{true};\)
for \( k = n \) to 3 do
choose \( v \neq v_1, v_2 \) such that \( \text{mark}(v) = \text{false}, \text{out}(v) = \text{true}, \)
\(\text{chords}(v) = 0;\)
\(v_k \leftarrow v; \text{mark}(v) \leftarrow \text{true};\)
// Let \( w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2 \) denote the boundary of \( G_{k-1}; \)
and let \( w_p, \ldots, w_q \) be the unmarked neighbors \( v_k; \)
\(\text{out}(w_i) \leftarrow \text{true} \) for all \( p < i < q;\)
update number of chords for \( w_i \) and its neighbors;
Computing Canonical Ordering

**Algorithm CO**

\[
\text{for all the } v \in V \text{ do} \\
\qquad \text{chords}(v) \leftarrow 0; \text{ out}(v) \leftarrow \text{false}; \text{ mark}(v) \leftarrow \text{false}; \\
\text{out}(v_1), \text{ out}(v_2), \text{ out}(v_n) \leftarrow \text{true}; \\
\text{for } k = n \text{ to } 3 \text{ do} \\
\qquad \text{choose } v \neq v_1, v_2 \text{ such that mark}(v) = \text{false}, \text{ out}(v) = \text{true}, \text{ chords}(v) = 0; \\
\qquad v_k \leftarrow v; \text{ mark}(v) \leftarrow \text{true}; \\
\qquad \text{// Let } w_1 = v_1, w_2, \ldots , w_{t-1}, w_t = v_2 \text{ denote the boundary of } G_{k-1}; \\
\qquad \text{and let } w_p, \ldots , w_q \text{ be the unmarked neighbors } v_k; \\
\qquad \text{out}(w_i) \leftarrow \text{true for all } p < i < q; \\
\qquad \text{update number of chords for } w_i \text{ and its neighbors;}
\]

- chord\((v)\) - number of chords adjacent to \(v\)
- mark\((v)\) = true iff vertex \(v\) was numbered
- out\((v)\)=true iff \(v\) is the outer vertex of current plane graph
Computing Canonical Ordering

Algorithm CO

\[
\text{forall the } v \in V \text{ do}
\]
\[
\text{chords}(v) \leftarrow 0; \text{out}(v) \leftarrow \text{false}; \text{mark}(v) \leftarrow \text{false};
\]
\[
\text{out}(v_1), \text{out}(v_2), \text{out}(v_n) \leftarrow \text{true};
\]
\[
\text{for } k = n \text{ to } 3 \text{ do}
\]
\[
\text{choose } v \neq v_1, v_2 \text{ such that mark}(v) = \text{false}, \text{out}(v) = \text{true},
\]
\[
\text{chords}(v) = 0;
\]
\[
v_k \leftarrow v; \text{mark}(v) \leftarrow \text{true};
\]
\[
// \text{Let } w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2 \text{ denote the boundary of } G_{k-1};
\]
\[
\text{and let } w_p, \ldots, w_q \text{ be the unmarked neighbors } v_k;
\]
\[
\text{out}(w_i) \leftarrow \text{true for all } p < i < q;
\]
\[
\text{update number of chords for } w_i \text{ and its neighbors};
\]

Lemma

Algorithm CO computes a canonical ordering of a graph in \(O(n)\) time.
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that
- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that

- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that

- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: \( G_{k-1} \) is drawn such that

- \( v_1 \) is on \((0, 0)\), \( v_2 \) is on \((2k - 6, 0)\)
- Boundary of \( G_{k-1} \) (minus edge \((v_1, v_2)\)) is drawn \( x \)-monotone
- Each edge of the boundary of \( G_{k-1} \) (minus edge \((v_1, v_2)\)) is drawn with slopes \( \pm 1 \)
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that

- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$

Overlaps! What could be the solution?
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that
- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
Algorithm constraints: $G_{k-1}$ is drawn such that

- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that
- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that

- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that

- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm

Algorithm constraints: $G_{k-1}$ is drawn such that

- $v_1$ is on $(0, 0)$, $v_2$ is on $(2k - 6, 0)$
- Boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn $x$-monotone
- Each edge of the boundary of $G_{k-1}$ (minus edge $(v_1, v_2)$) is drawn with slopes $\pm 1$
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm

$L(10)$
De Fraysseix Pach Pollack (Shift) Algorithm

\[ L(11) \]
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm
De Fraysseix Pach Pollack (Shift) Algorithm

\[ (0, 0) \rightarrow (2n - 4, 0) \rightarrow (n - 2, n - 2) \]

\[ (0, 0) \rightarrow (2n - 4, 0) \rightarrow (n - 2, n - 2) \]
De Fraysseix Pach Pollack (Shift) Algorithm

$V_k$

$G_{k-1}$
De Fraysseix Pach Pollack (Shift) Algorithm

Covered vertices

$G_{k-1}$

$U_k$
De Fraysseix Pach Pollack (Shift) Algorithm

- Each internal vertex is covered exactly once
- Coverence relation defines a tree in $G$
- But a forest in $G_i$, $1 \leq i \leq n-1$
De Fraysseix Pach Pollack (Shift) Algorithm

- Each internal vertex is covered exactly once
- Coverence relation defines a tree in \( G \)
- But a forest in \( G_i, 1 \leq i \leq n-1 \)
De Fraysseix Pach Pollack (Shift) Algorithm

- Each internal vertex is covered exactly once
- Coverence relation defines a tree in $G$
- But a forest in $G_i$, $1 \leq i \leq n-1$
De Fraysseix Pach Pollack (Shift) Algorithm

- Each internal vertex is covered exactly once
- Coverence relation defines a tree in $G$
- But a forest in $G_i$, $1 \leq i \leq n-1$
De Fraysseix Pach Pollack (Shift) Algorithm

- Each internal vertex is covered exactly once
- Coverage relation defines a tree in $G$
- But a forest in $G_i$, $1 \leq i \leq n-1$
De Fraysseix Pach Pollack (Shift) Algorithm

- Each internal vertex is covered exactly once
- Coverage relation defines a tree in $G$
- But a forest in $G_i$, $1 \leq i \leq n-1$

**Lemma**

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.
Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.
Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.

Proof

- The proof is by induction on $i$, i.e. we consider $G_3, \ldots, G_n$.  

Lemma

Let \(0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}\). If we shift \(L(w_i)\) by \(\delta_i\) to the right, we get a planar straight line drawing.

Proof

- The proof is by induction on \(i\), i.e. we consider \(G_3, \ldots, G_n\).
- Assume that this is true for \(G_{k-1}\).
### Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.

### Proof

- The proof is by induction on $i$, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for $G_{k-1}$.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of $G_k$. 

---

**De Fraysseix Pach Pollack (Shift) Algorithm**

---

*Institut für Theoretische Informatik*

*Lehrstuhl Algorithmik I*
De Fraysseix Pach Pollack (Shift) Algorithm

**Lemma**

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.

**Proof**

- The proof is by induction on $i$, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for $G_{k-1}$.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of $G_k$.
- Let $\delta(w_1) \leq \cdots \leq \delta(w_p) \leq \delta(v_k) \leq \delta(w_q) \leq \cdots \leq \delta(w_t)$. 

De Fraysseix Pach Pollack (Shift) Algorithm

**Lemma**

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.

**Proof**

- The proof is by induction on $i$, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for $G_{k-1}$.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of $G_k$.
- Let $\delta(w_1) \leq \cdots \leq \delta(w_p) \leq \delta(v_k) \leq \delta(w_q) \leq \cdots \leq \delta(w_t)$.
- We set $\delta'(w_i) = \delta(w_i)$ for $1 \leq i \leq p$,
- $\delta'(w_i) = \delta(w_i) + 1$ for $p + 1 \leq i \leq q - 1$
- $\delta'(w_i) = \delta(w_i) + 2$ for $q \leq i \leq t$. 
Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.

Proof

- The proof is by induction on $i$, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for $G_{k-1}$.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of $G_k$.
- Let $\delta(w_1) \leq \cdots \leq \delta(w_p) \leq \delta(v_k) \leq \delta(w_q) \leq \cdots \leq \delta(w_t)$.
- We set $\delta'(w_i) = \delta(w_i)$ for $1 \leq i \leq p$,
  $\delta'(w_i) = \delta(w_i) + 1$ for $p + 1 \leq i \leq q - 1$
  $\delta'(w_i) = \delta(w_i) + 2$ for $q \leq i \leq t$.
- By induction hypothesis we can move $w_1, \ldots, w_t$ by $\delta(w_1)' \ldots \delta(w_t)'$, respectively.
Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift $L(w_i)$ by $\delta_i$ to the right, we get a planar straight line drawing.

Proof

- The proof is by induction on $i$, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for $G_{k-1}$.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of $G_k$.
- Let $\delta(w_1) \leq \cdots \leq \delta(w_p) \leq \delta(v_k) \leq \delta(w_q) \leq \cdots \leq \delta(w_t)$.
- We set $\delta'(w_i) = \delta(w_i)$ for $1 \leq i \leq p$.
- $\delta'(w_i) = \delta(w_i) + 1$ for $p + 1 \leq i \leq q - 1$
- $\delta'(w_i) = \delta(w_i) + 2$ for $q \leq i \leq t$.
- By induction hypothesis we can move $w_1, \ldots, w_t$ by $\delta(w_1)' \ldots \delta(w_t)'$, respectively.
- We can complete the drawing by placing $v_k$. 
Algorithm Shift

Let \( v_1, \ldots, v_n \) be a canonical ordering of \( G \)

\[
\text{for } i = 1 \text{ to } n \text{ do }
\]
\[
L(v_i) \leftarrow \{v_i\};
\]
\[
P(v_1) \leftarrow (0, 0); \ P(v_2) \leftarrow (2, 0); \ P(v_3) \leftarrow (1, 1);
\]

\[
\text{for } i = 4 \text{ to } n \text{ do }
\]
\[
\text{Let } w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2 \text{ denote the boundary of } G_{i-1};
\]
\[
\text{and let } w_p, \ldots, w_q \text{ be the neighbors } v_i;
\]
\[
\text{for } \forall v \in \bigcup_{j=p+1}^{q-1} L(w_j) \text{ do }
\]
\[
x(v) \leftarrow x(v) + 1;
\]
\[
\text{for } \forall v \in \bigcup_{j=q}^{t} L(w_j) \text{ do }
\]
\[
x(v) \leftarrow x(v) + 2;
\]
\[
P(v_i) \leftarrow \text{intersection of } +1 \text{ and } -1 \text{ edges from } P(w_p) \text{ and } P(w_q);
\]
\[
L(v_i) = \bigcup_{j=p+1}^{q-1} L(w_j) \cup \{v_i\};
\]
Linear Time Implementation of Shift Algorithm
Linear Time Implementation of Shift Algorithm
Linear Time Implementation of Shift Algorithm

\[ x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p)) \] (1)

\[ y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p)) \] (2)

\[ x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p)) \] (3)
Linear Time Implementation of Shift Algorithm

- If we know the \( y \)-coordinates of \( w_p \) and \( w_q \) and the difference \( x(w_p) - x(w_q) \), we can compute the relative distance of \( v_k \) and \( w_p \).
- In the binary tree which we construct we keep the relative \( x \)-distance of each node from its parent.
Linear Time Implementation of Shift Algorithm

- If we know the $y$-coordinates of $w_p$ and $w_q$ and the difference $x(w_p) - x(w_q)$, we can compute the relative distance of $v_k$ and $w_p$.
- In the binary tree which we construct we keep the relative $x$-distance of each node from its parent.
If we know the $y$-coordinates of $w_p$ and $w_q$ and the difference $x(w_p) - x(w_q)$, we can compute the relative distance of $v_k$ and $w_p$.

In the binary tree which we construct we keep the relative $x$-distance of each node from its parent.

- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)
If we know the $y$-coordinates of $w_p$ and $w_q$ and the difference $x(w_p) - x(w_q)$, we can compute the relative distance of $v_k$ and $w_p$.

In the binary tree which we construct we keep the relative $x$-distance of each node from its parent.

$$\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$$

Calculate $\Delta_x(v_k)$ by eq. (3)

Calculate $y(v_k)$ by eq. (2)
If we know the $y$-coordinates of $w_p$ and $w_q$ and the difference $x(w_p) - x(w_q)$, we can compute the relative distance of $v_k$ and $w_p$.

In the binary tree which we construct we keep the relative $x$-distance of each node from its parent.

$\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$

Calculate $\Delta_x(v_k)$ by eq. (3)

Calculate $y(v_k)$ by eq. (2)
If we know the $y$-coordinates of $w_p$ and $w_q$ and the difference $x(w_p) - x(w_q)$, we can compute the relative distance of $v_k$ and $w_p$.

In the binary tree which we construct we keep the relative $x$-distance of each node from its parent.

$\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$

Calculate $\Delta_x(v_k)$ by eq. (3)

Calculate $y(v_k)$ by eq. (2)
Linear Time Implementation of Shift Algorithm

- If we know the $y$-coordinates of $w_p$ and $w_q$ and the difference $x(w_p) - x(w_q)$, we can compute the relative distance of $v_k$ and $w_p$.
- In the binary tree which we construct we keep the relative $x$-distance of each node from its parent.
- $\Delta x(w_p, w_q) = \Delta x(w_{p+1}) + \cdots + \Delta x(w_q)$
- Calculate $\Delta x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)
Linear Time Implementation of Shift Algorithm

- If we know the \(y\)-coordinates of \(w_p\) and \(w_q\) and the difference \(x(w_p) - x(w_q)\), we can compute the relative distance of \(v_k\) and \(w_p\).
- In the binary tree which we construct we keep the relative \(x\)-distance of each node from its parent.
- \(\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)\)
- Calculate \(\Delta_x(v_k)\) by eq. (3)
- Calculate \(y(v_k)\) by eq. (2)