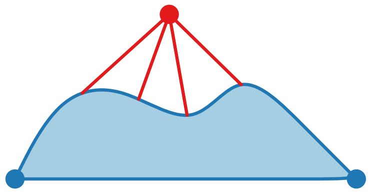
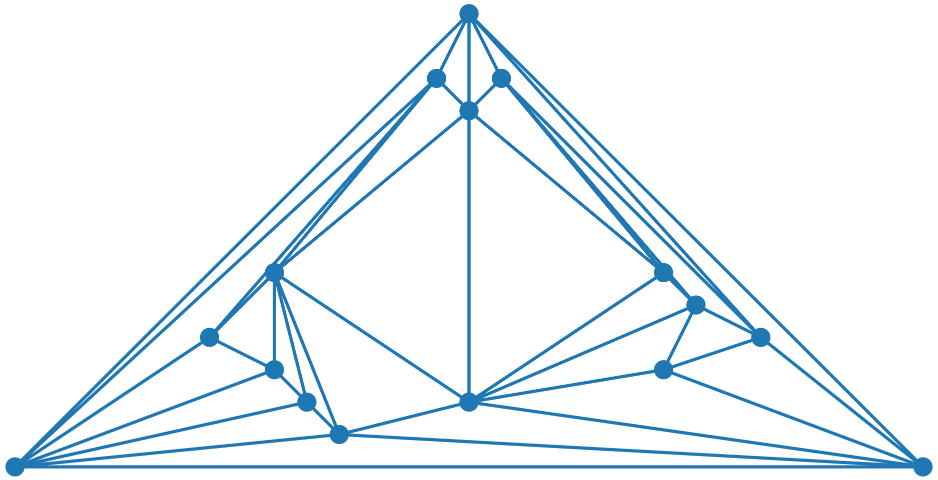
Visualisation of graphs

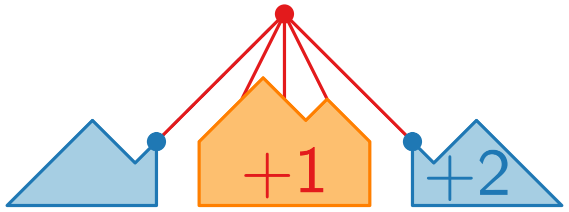
Planar straight-line drawings

Canonical order & shift method

Jonathan Klawitter · Summer semester 2020







Motivation

- So far we looked at planar and straight-line drawings of trees and series-parallel graphs.

Motivation

- So far we looked at planar and straight-line drawings of trees and series-parallel graphs.
- Why straight-line? Why planar?

Motivation

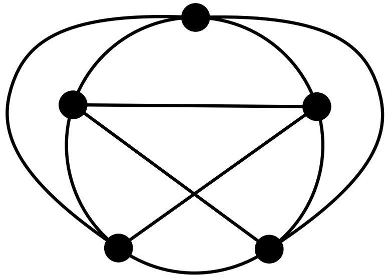
- So far we looked at planar and straight-line drawings of trees and series-parallel graphs.
- Why straight-line? Why planar?
- Bennett, Ryall, Spaltzeholz and Gooch, 2007
  "The Aesthetics of Graph Visualization"

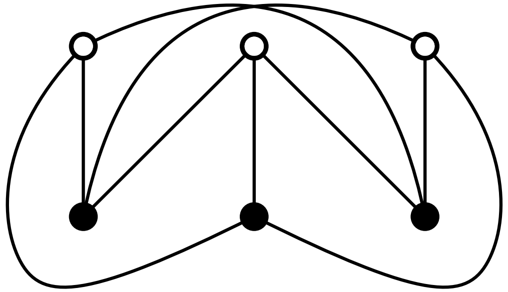
3.2. Edge Placement Heuristics

By far the most agreed-upon edge placement heuristic is to *minimize the number of edge crossings* in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, based on perceptual principles, it is beneficial to *minimize the number of edge bends* within a graph [Pur02, TR05, TBB88]. Edge bends make edges more difficult to follow because an edge with a sharp bend is more likely to be perceived as two separate objects. This leads to the heuristic of *keeping edge bends uniform* with respect to the bend's position on the edge and its angle [TR05]. If an edge must be bent to satisfy other aesthetic criteria, the angle of the bend should be as little as possible, and the bend placement should evenly divide the edge.

Planar graphs

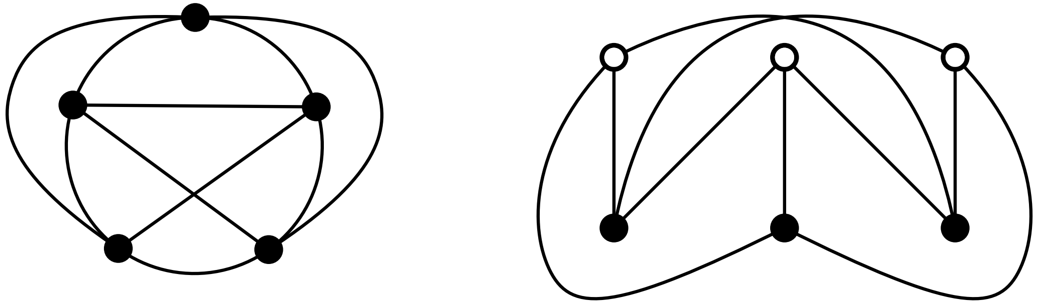
- **Characterisation:** A graph is planar iff it contains neither a K_5 nor a $K_{3,3}$ minor.
  [Kuratowski 1930]





Planar graphs

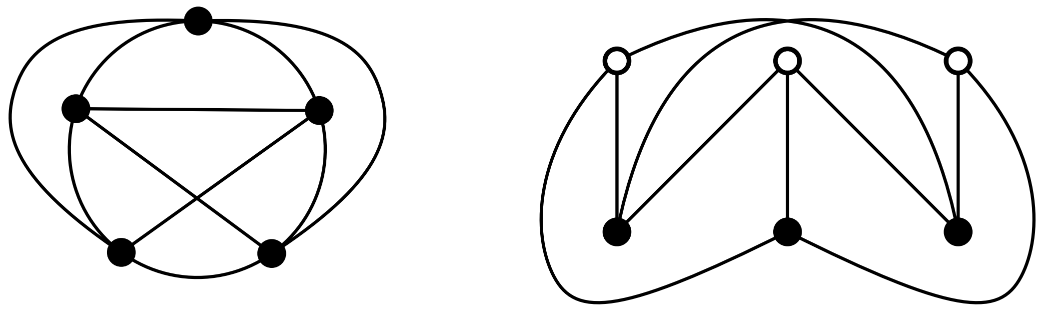
- **Characterisation:** A graph is planar iff it contains neither a K_5 nor a $K_{3,3}$ minor. [Kuratowski 1930]



- **Recognition:** For a graph G with n vertices, there is an $\mathcal{O}(n)$ time algorithm to test if G is planar. [Hopcroft & Tarjan 1974]
  - Also computes an embedding in $\mathcal{O}(n)$.

Planar graphs

- **Characterisation:** A graph is planar iff it contains neither a K_5 nor a $K_{3,3}$ minor. [Kuratowski 1930]

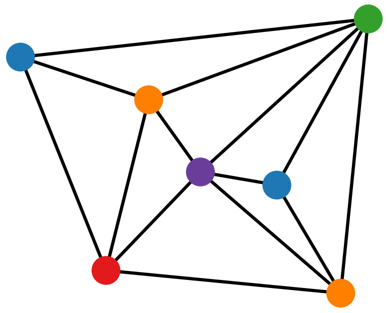


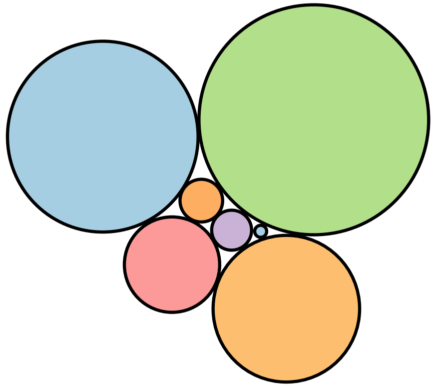
- **Recognition:** For a graph G with n vertices, there is an $\mathcal{O}(n)$ time algorithm to test if G is planar. [Hopcroft & Tarjan 1974]
  - Also computes an embedding in $\mathcal{O}(n)$.

- **Straight-line drawing:** Every planar graph has an embedding where the edges are straight-line segments. [Wagner 1936, Fáry 1948, Stein 1951]
  - The algorithms implied by this theory produce drawings with area not bounded by any polynomial on n.

Planar graphs

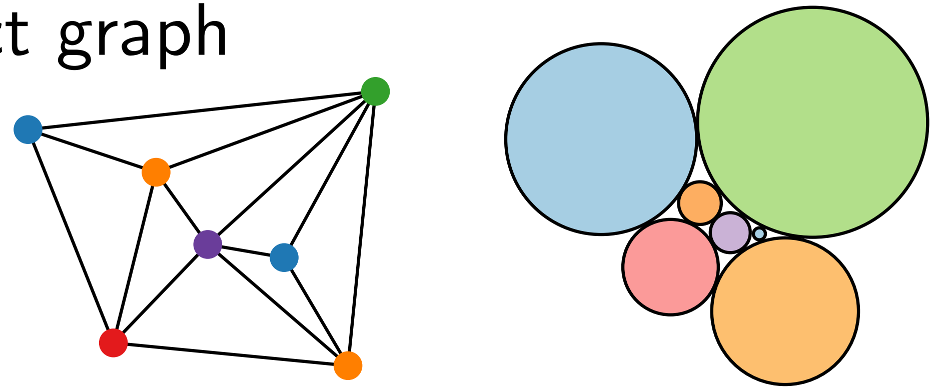
- **Coin graph:** Every planar graph is a circle contact graph (implies straight-line drawing). [Koebe 1936]





Planar graphs

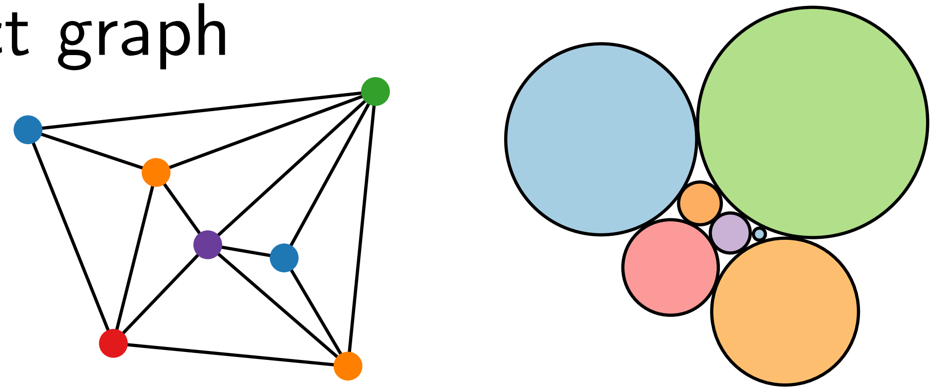
- **Coin graph:** Every planar graph is a circle contact graph (implies straight-line drawing). [Koebe 1936]



- Every 3-connected planar graph has an embedding with convex polygons as its faces (i.e., implies straight lines). [Tutte 1963: How to draw a graph]
  - Idea: Place vertices in the barycentre of neighbours.
  - Drawback: Requires large grids.

Planar graphs

- **Coin graph:** Every planar graph is a circle contact graph (implies straight-line drawing). [Koebe 1936]



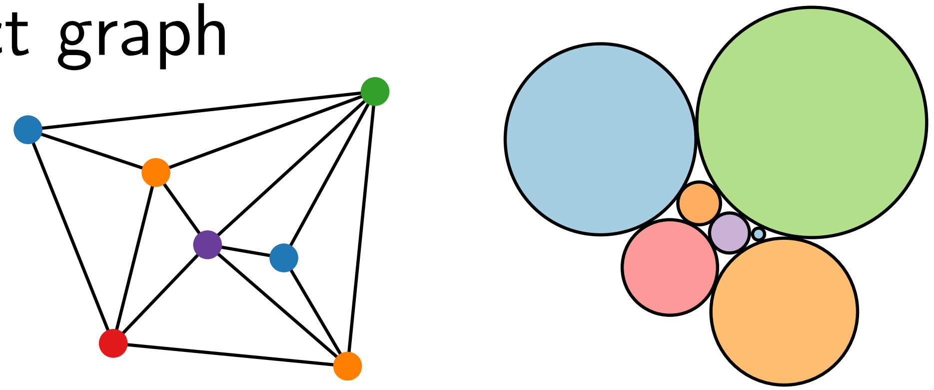
- Every 3-connected planar graph has an embedding with convex polygons as its faces (i.e., implies straight lines). [Tutte 1963: How to draw a graph]
  - Idea: Place vertices in the barycentre of neighbours.
  - Drawback: Requires large grids.

with planar embedding

- We focus on **triangulations**:
  - A *plane (inner) triangulation* is a plane graph where every (inner) face is a triangle.
  - Every plane graph is subgraph of a plane triangulation.

Planar graphs

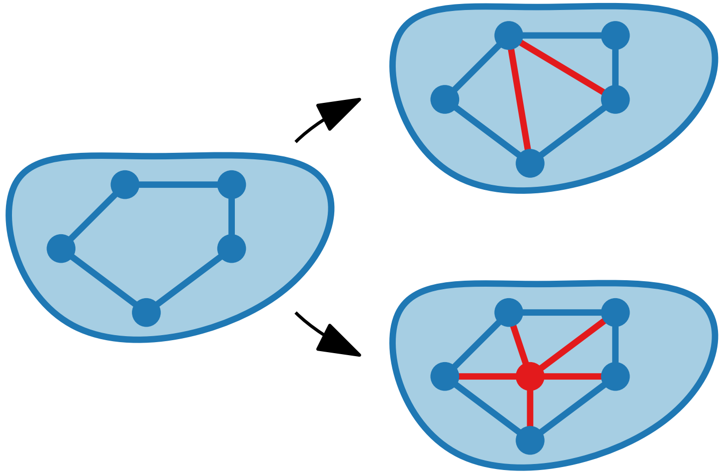
- **Coin graph:** Every planar graph is a circle contact graph (implies straight-line drawing). [Koebe 1936]



- Every 3-connected planar graph has an embedding with convex polygons as its faces (i.e., implies straight lines). [Tutte 1963: How to draw a graph]
  - Idea: Place vertices in the barycentre of neighbours.
  - Drawback: Requires large grids.

- We focus on **triangulations**:
  - A *plane (inner) triangulation* is a plane graph where every (inner) face is a triangle.
  - Every plane graph is subgraph of a plane triangulation.

with planar embedding



Planar straight-line drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

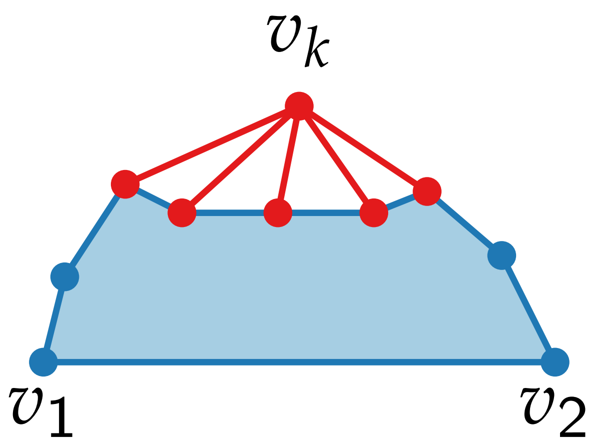
Theorem. [Schnyder '90] Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

Planar straight-line drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Idea.

- Start with singe edge (v_1, v_2). Let this be G_2.
- To obtain G_{i+1}, add v_{i+1} to G_i so that neighbours of v_{i+1} are on the outer face of G_i.
- Neighbours of v_{i+1} in G_i have to form path of length at least two.



Theorem. [Schnyder '90] Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

Canonical order – definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices.
An order $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

Canonical order – definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices. An order $\pi = (v_1, v_2, \dots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

- (C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

Canonical order – definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices. An order $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

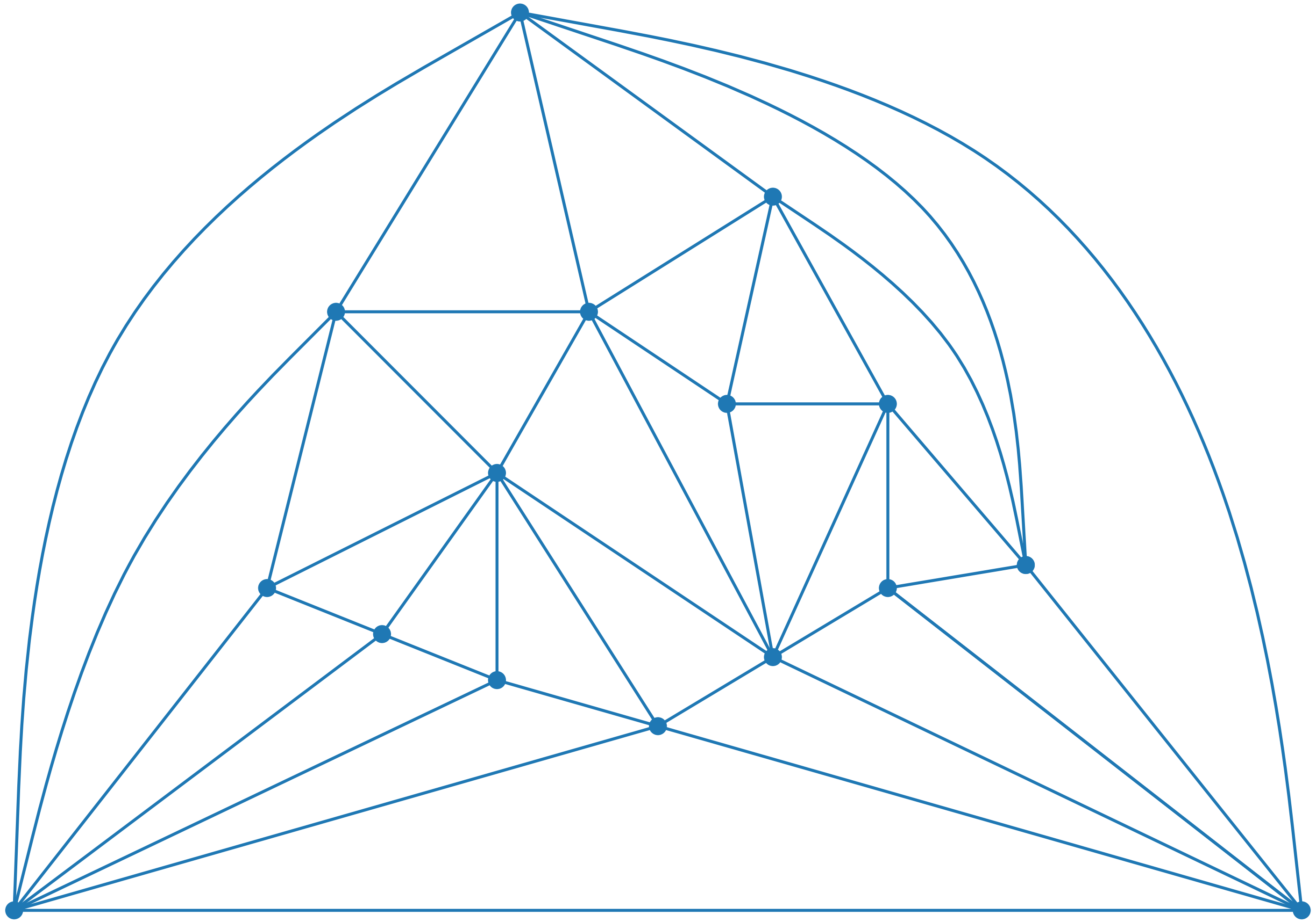
- (C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.
- (C2) Edge (v_1, v_2) belongs to the outer face of G_k.

Canonical order – definition

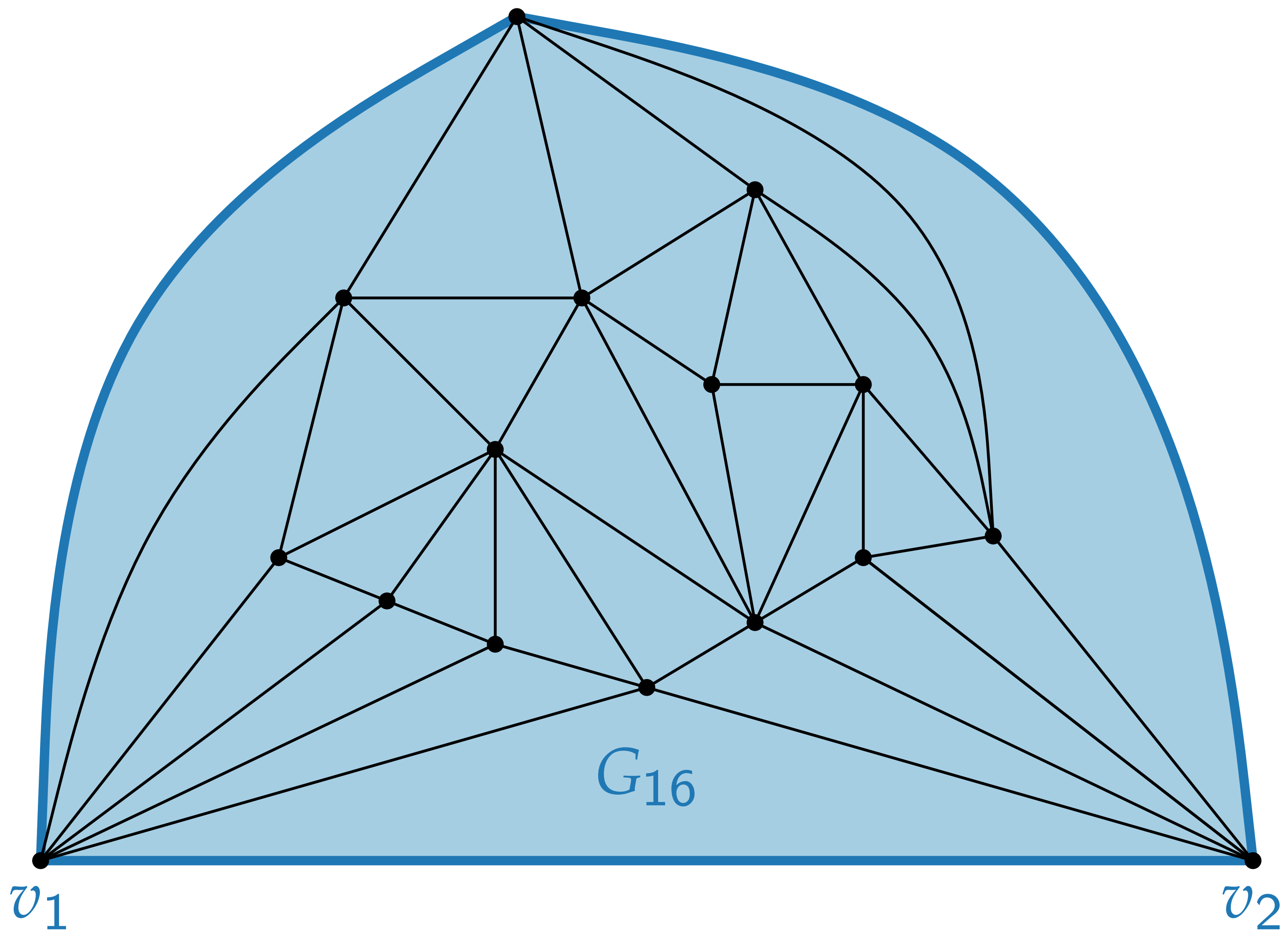
Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices. An order $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

- (C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected 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.

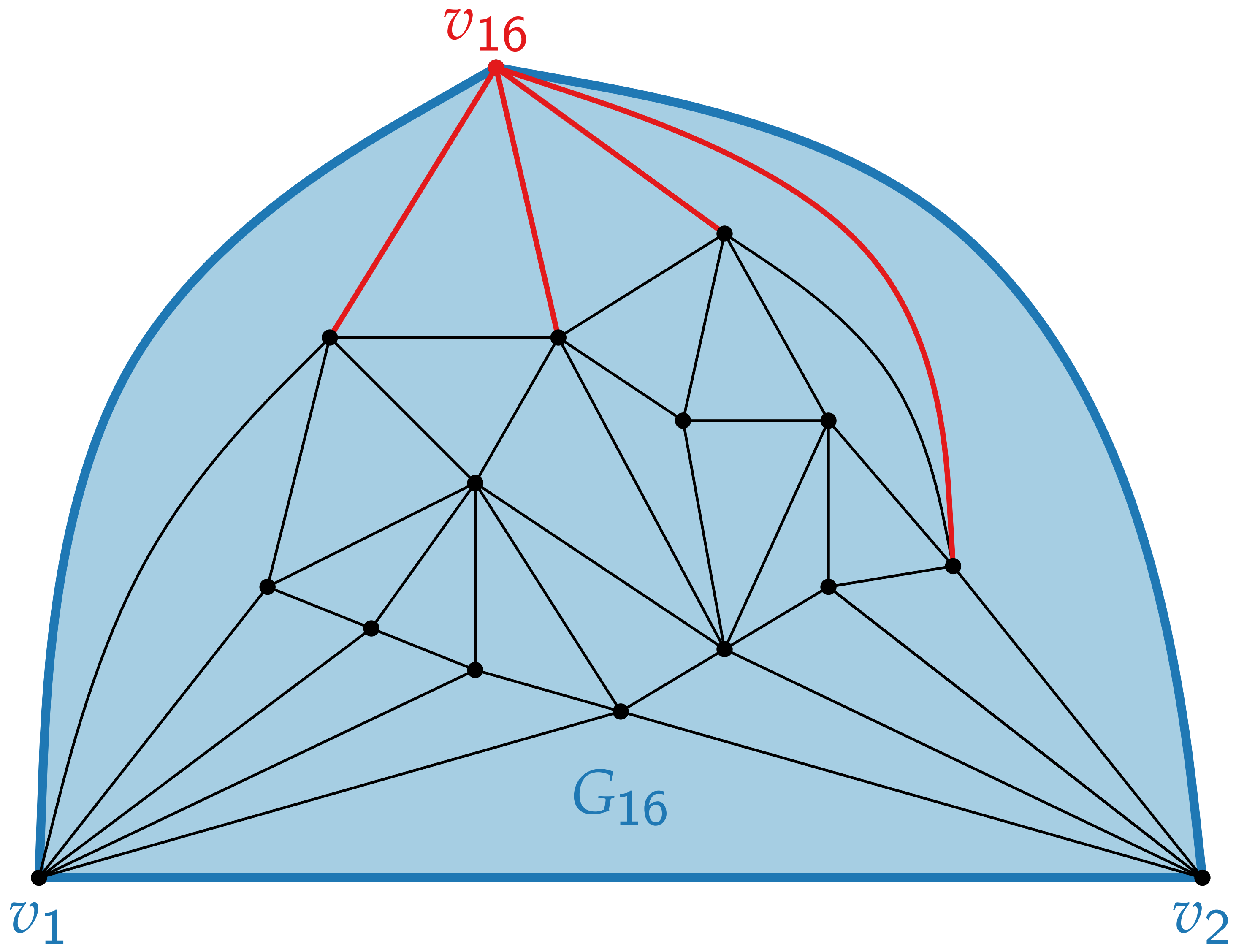
Canonical order – example



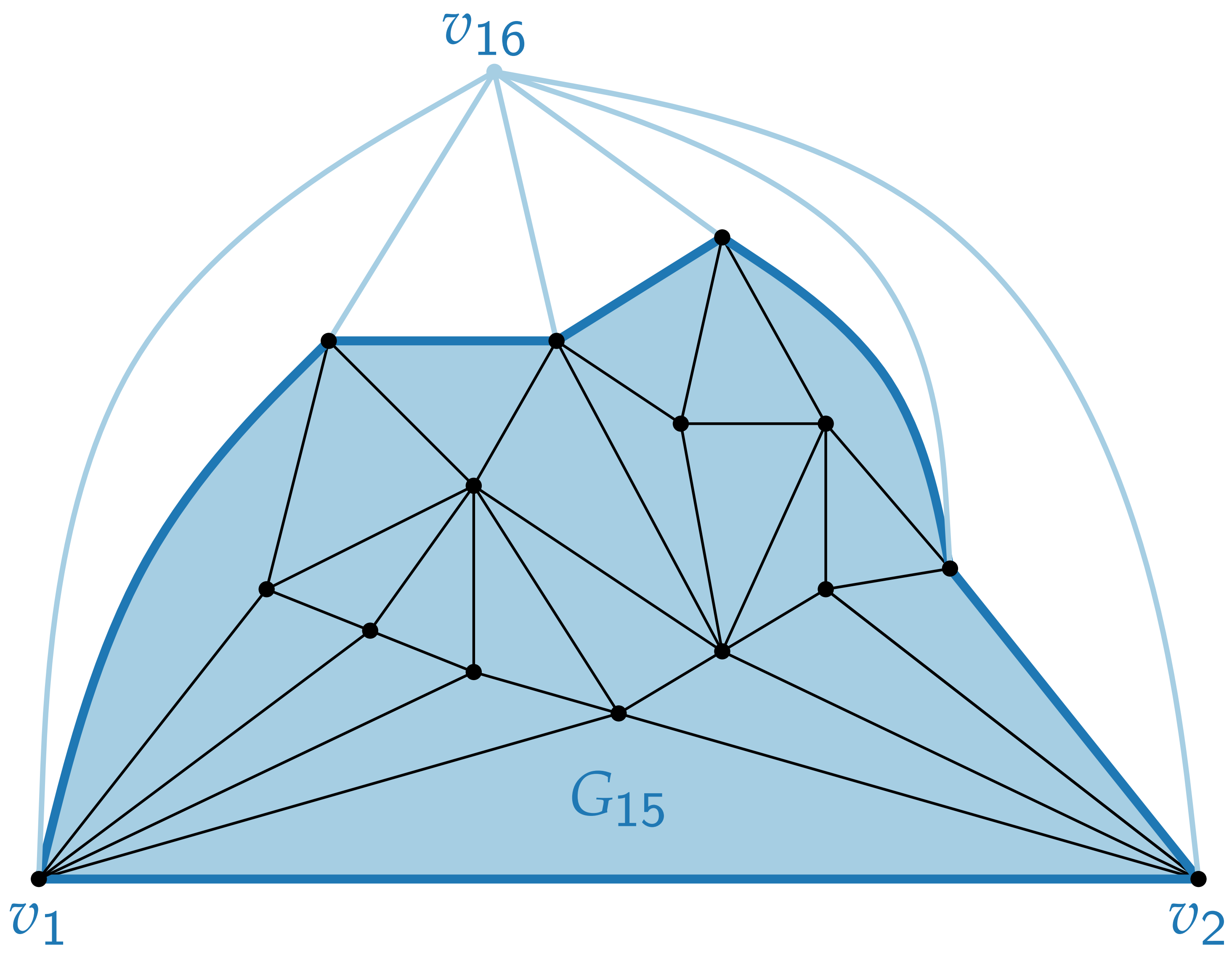
Canonical order – example



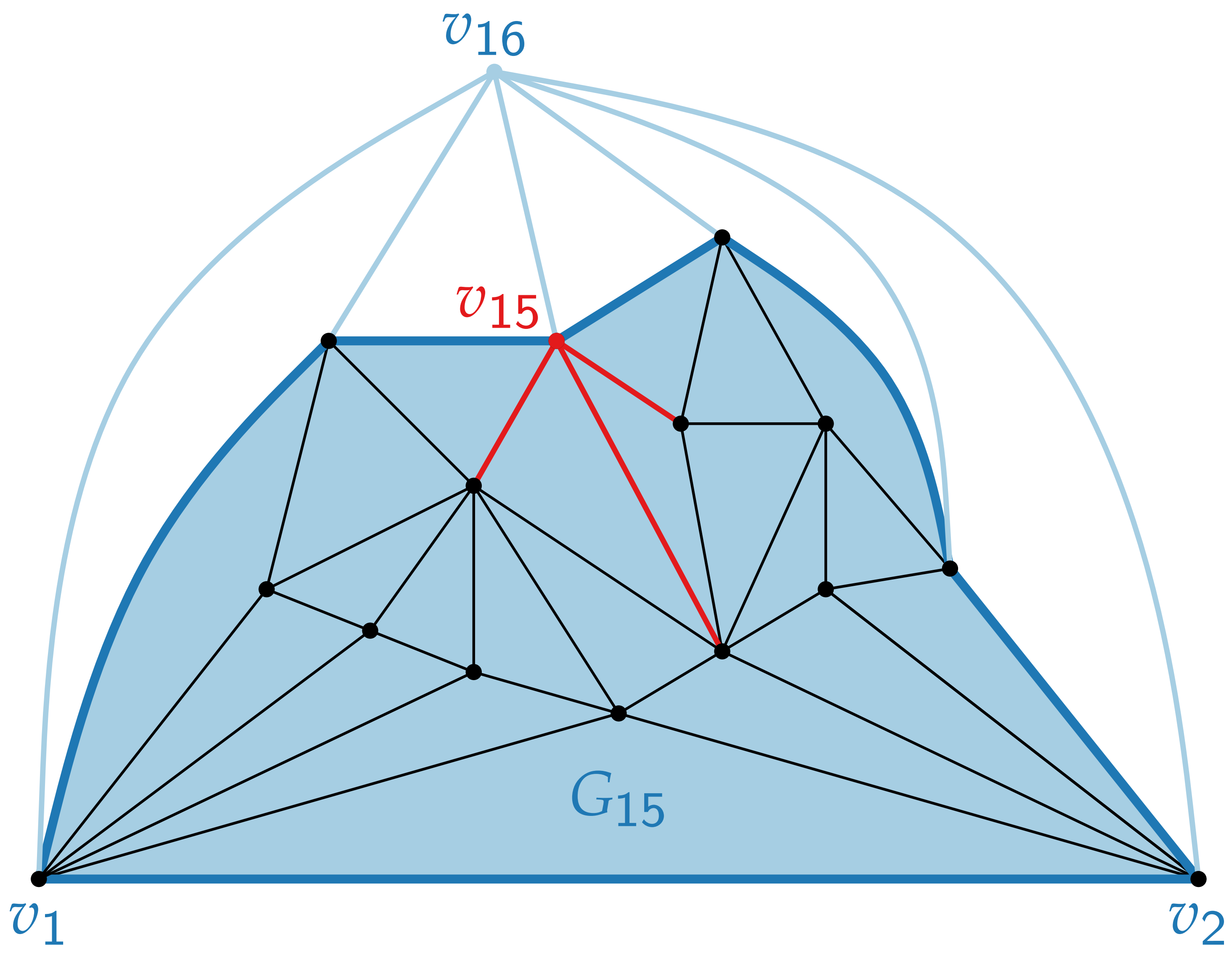
Canonical order – example



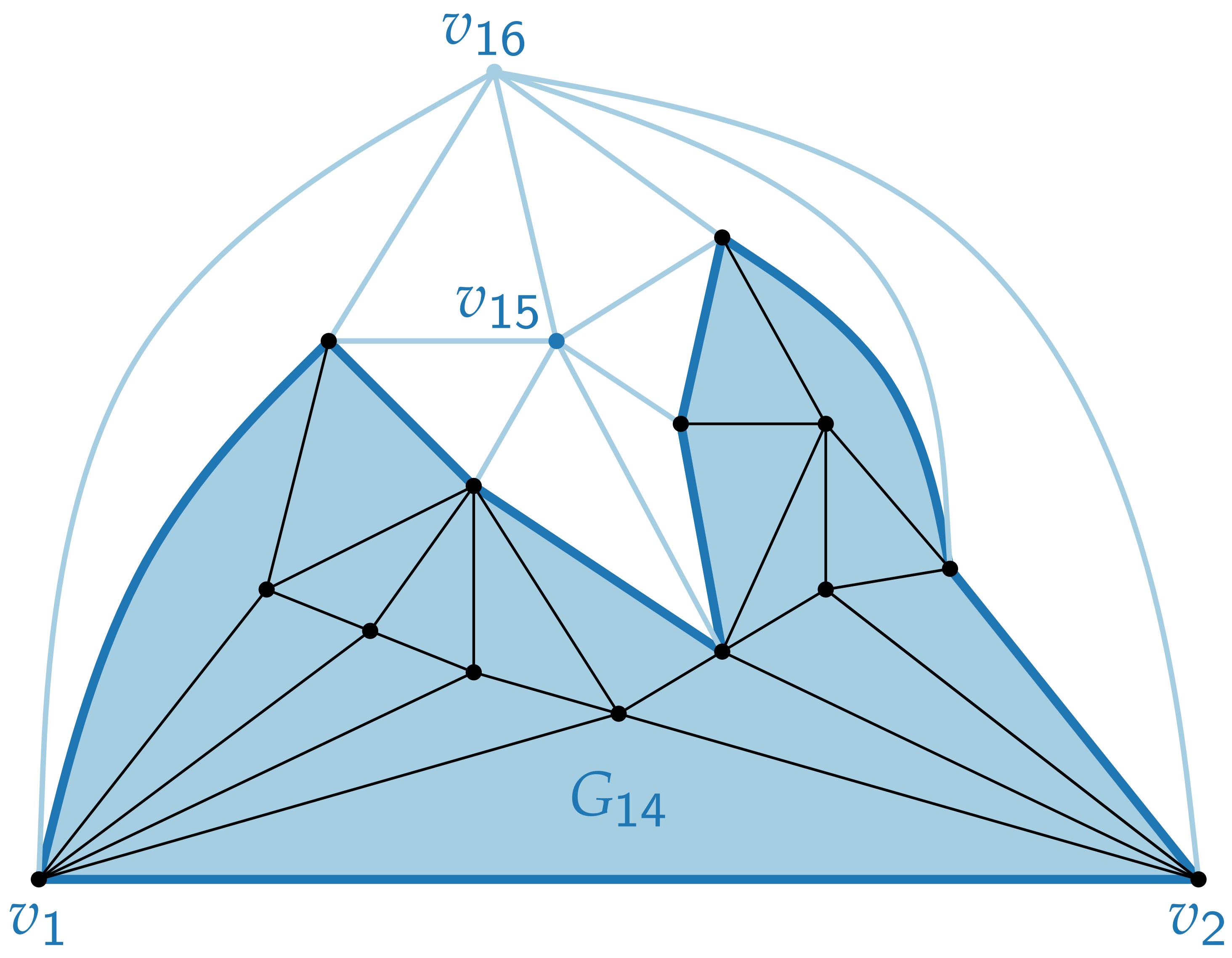
Canonical order – example



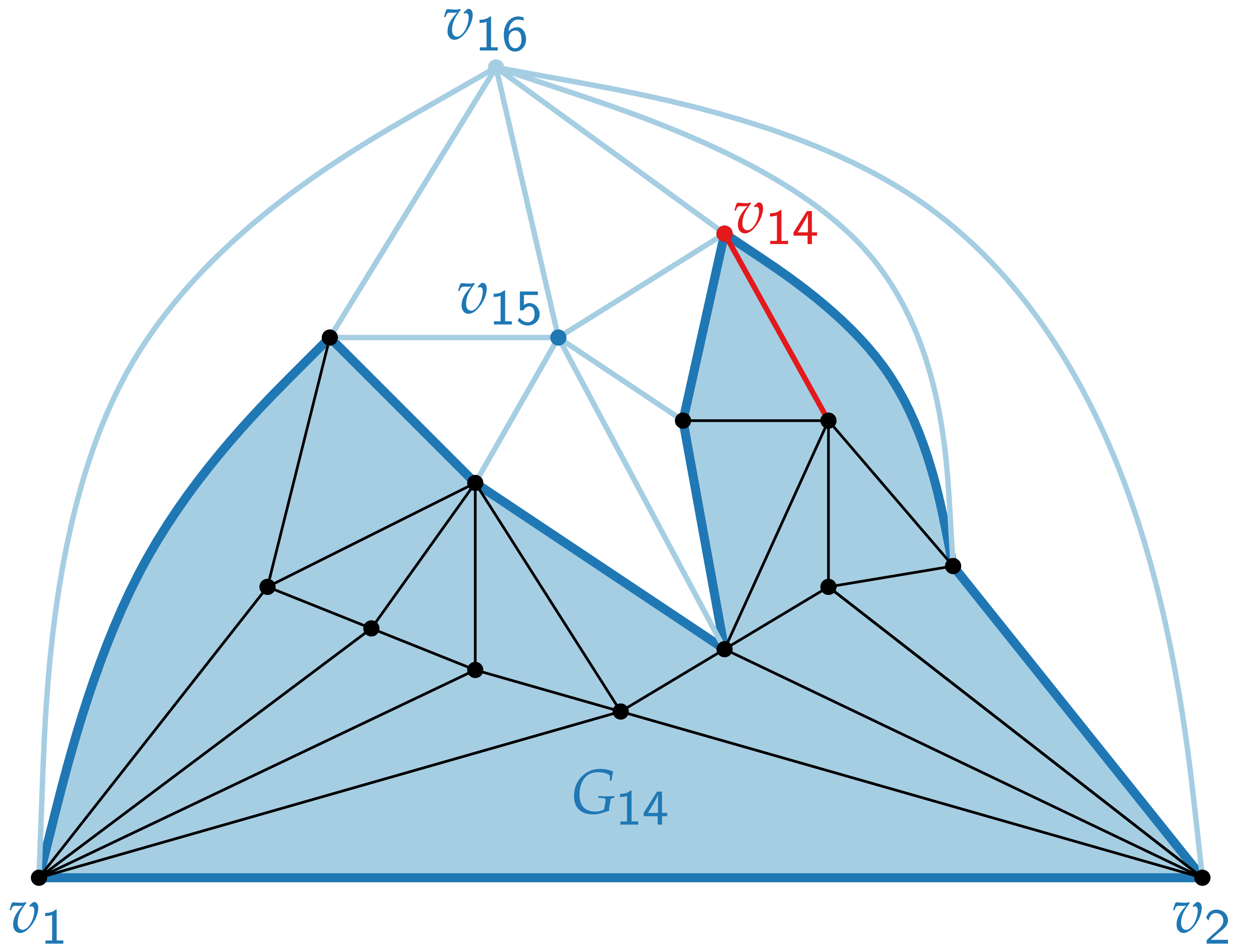
Canonical order – example



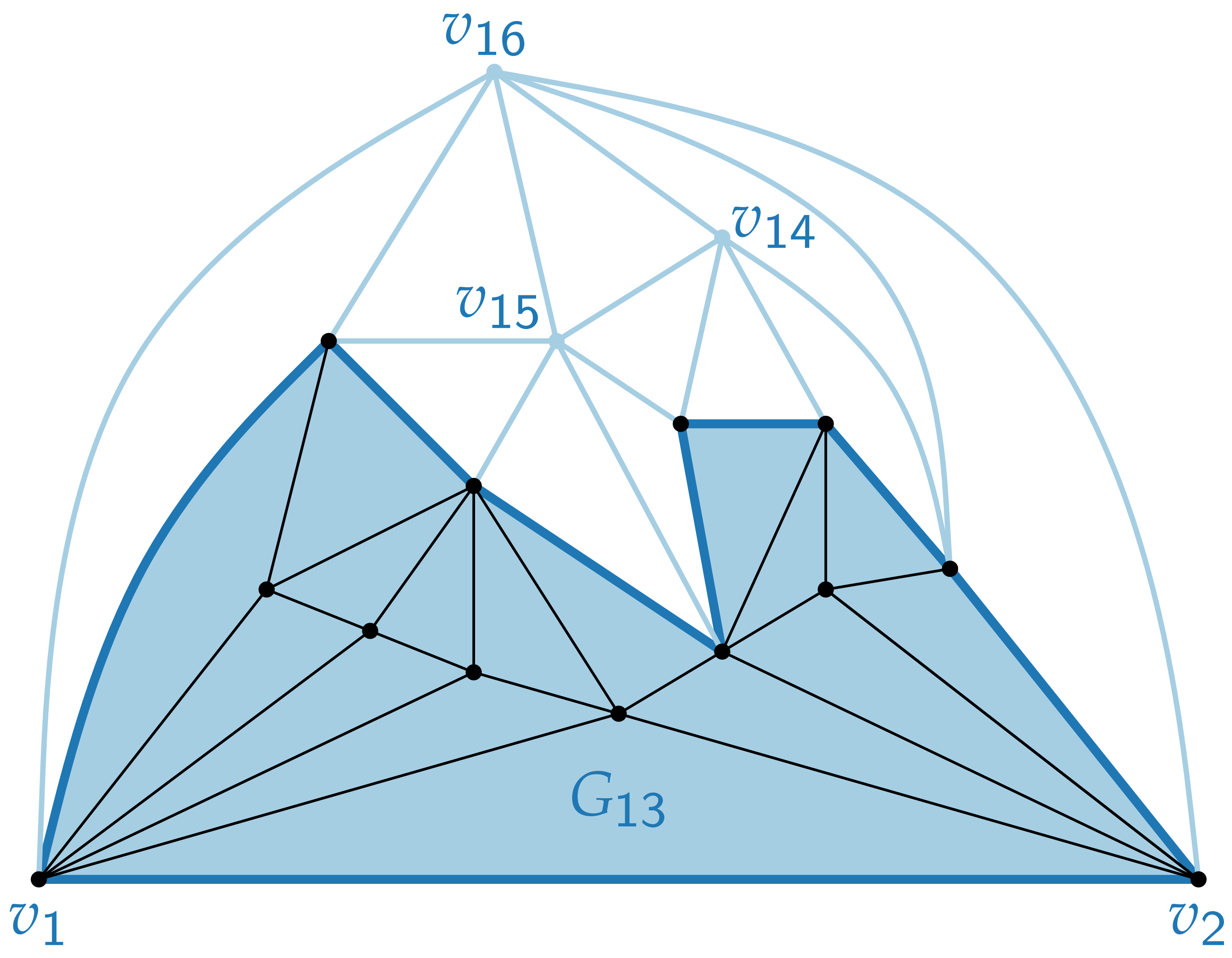
Canonical order – example



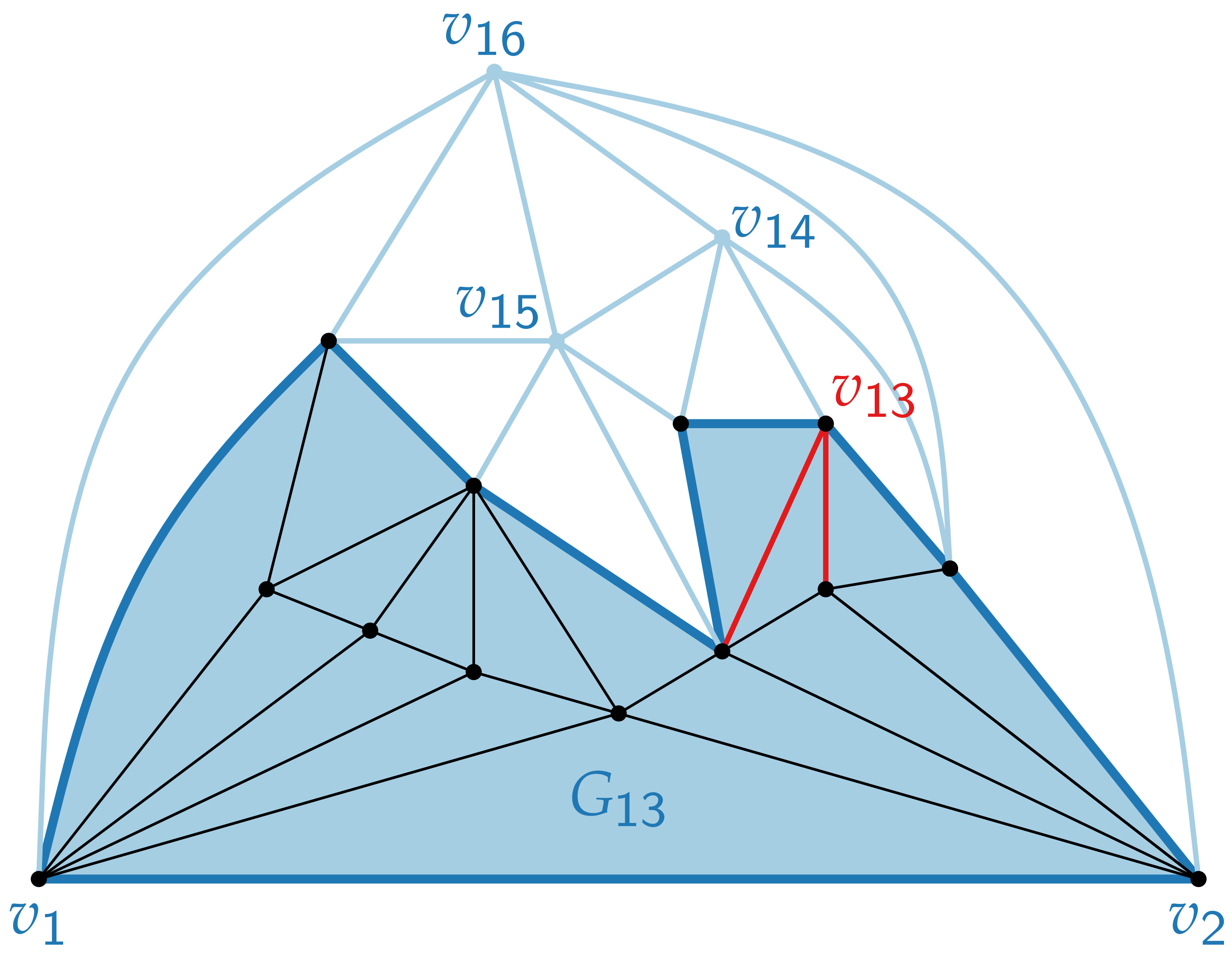
Canonical order – example



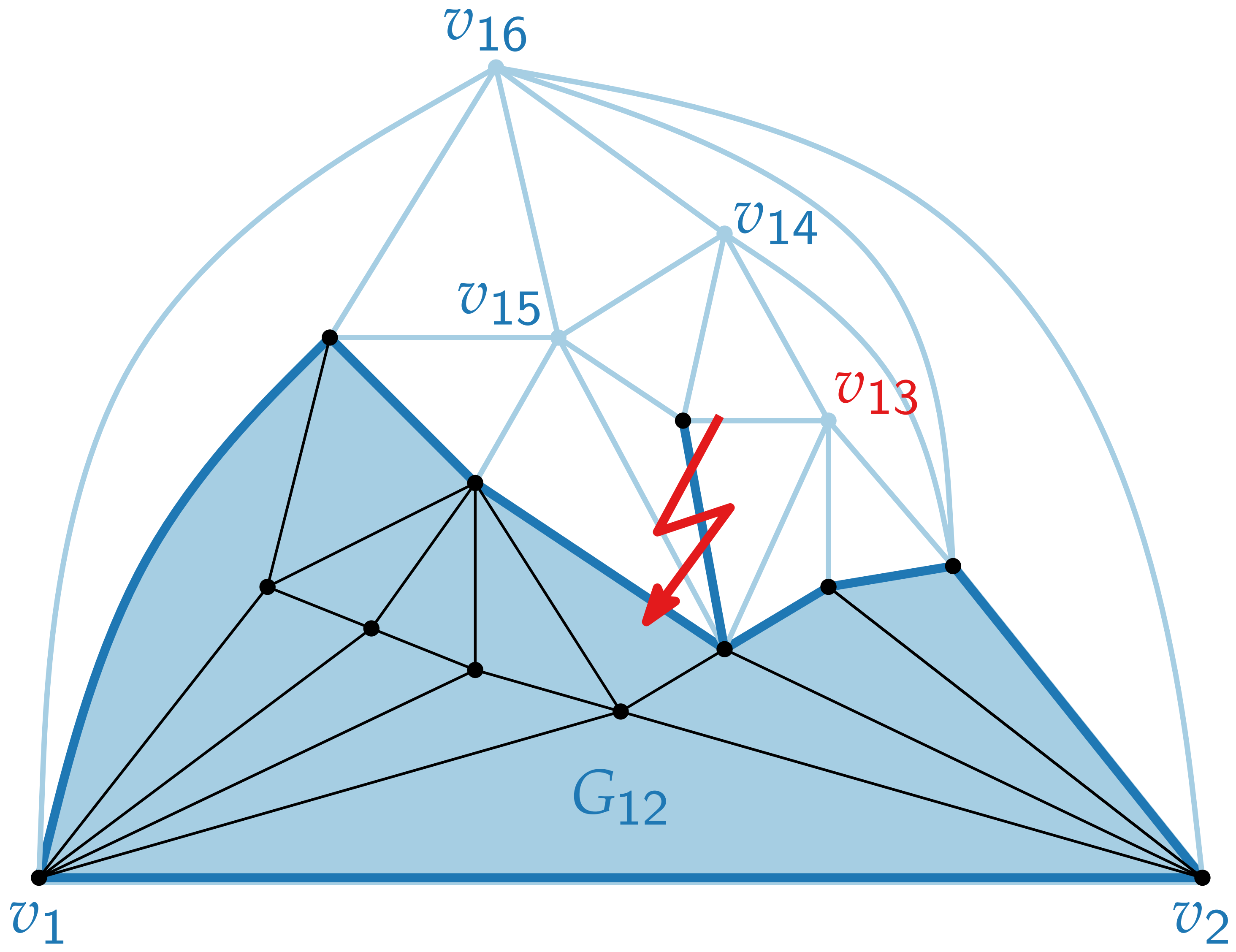
Canonical order – example



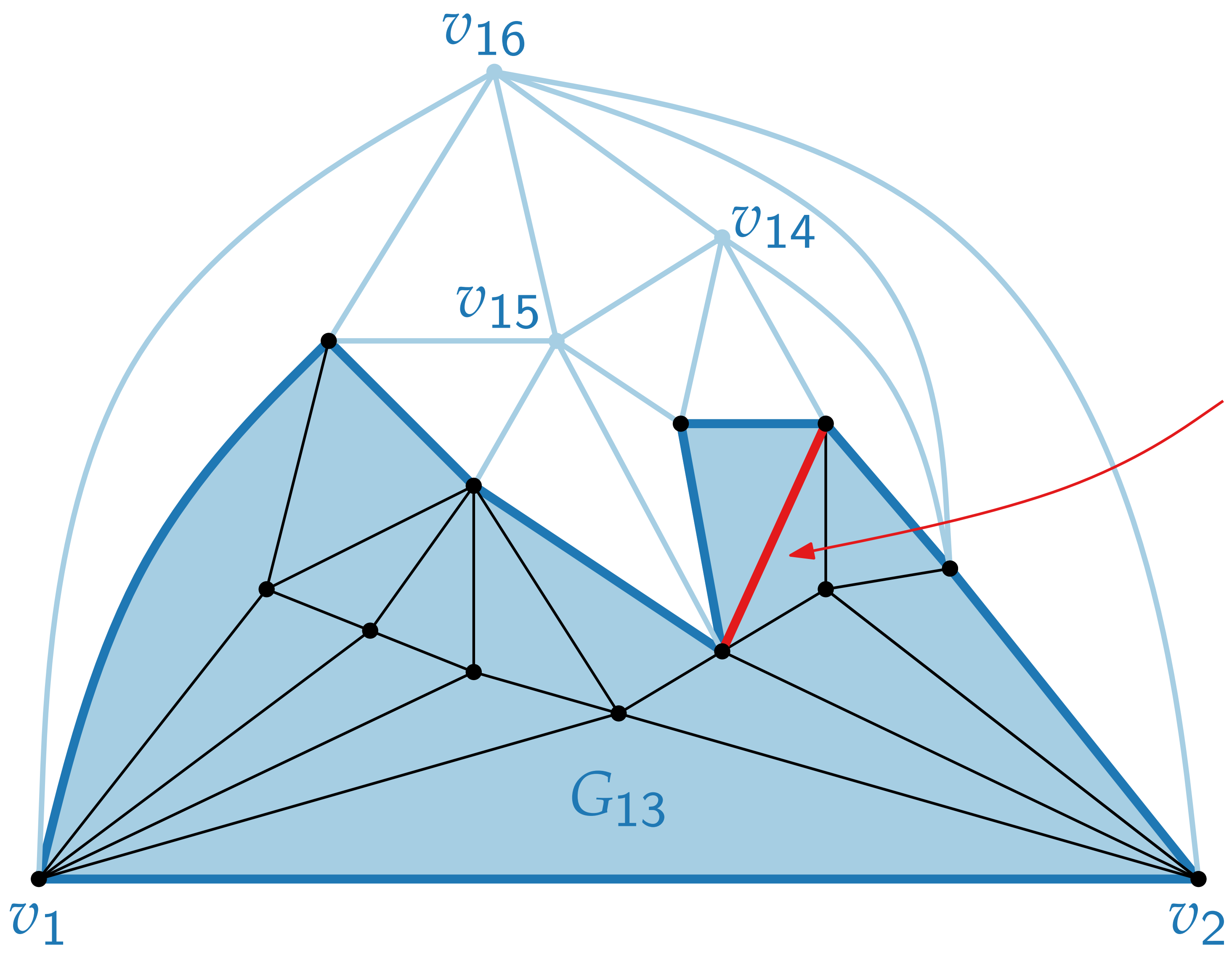
Canonical order – example



Canonical order – example

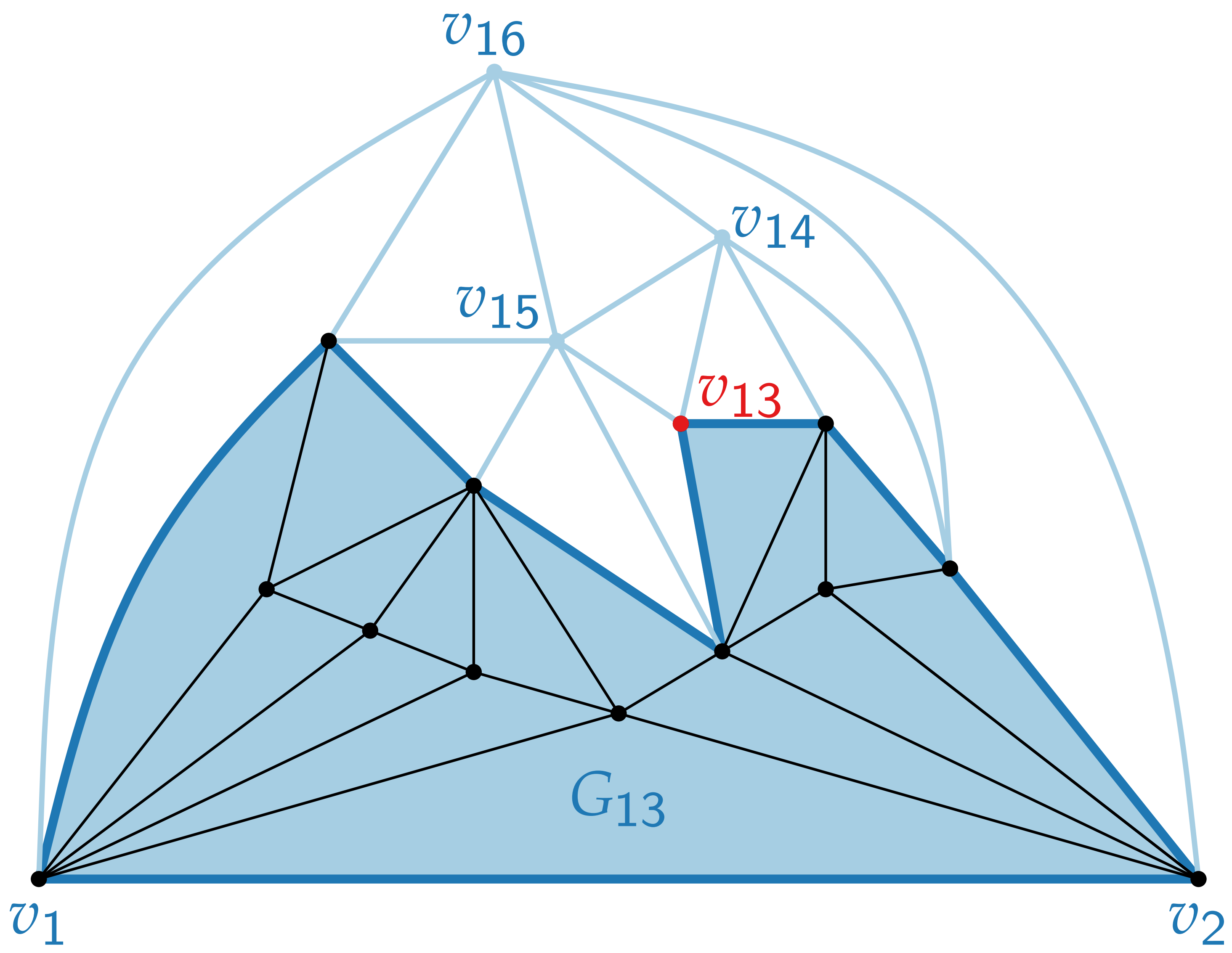


Canonical order – example

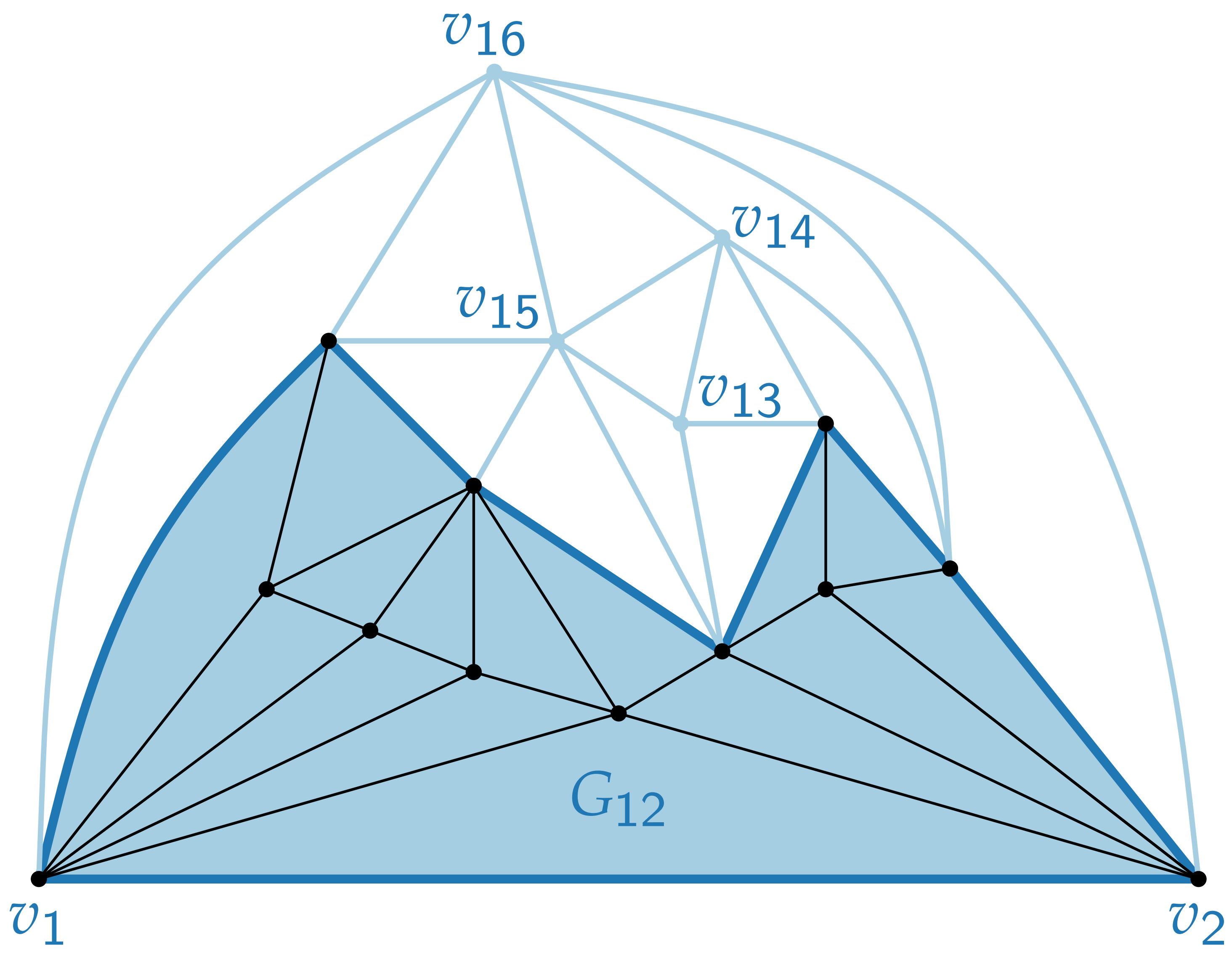


chord
edge joining two nonadjacent vertices in a cycle

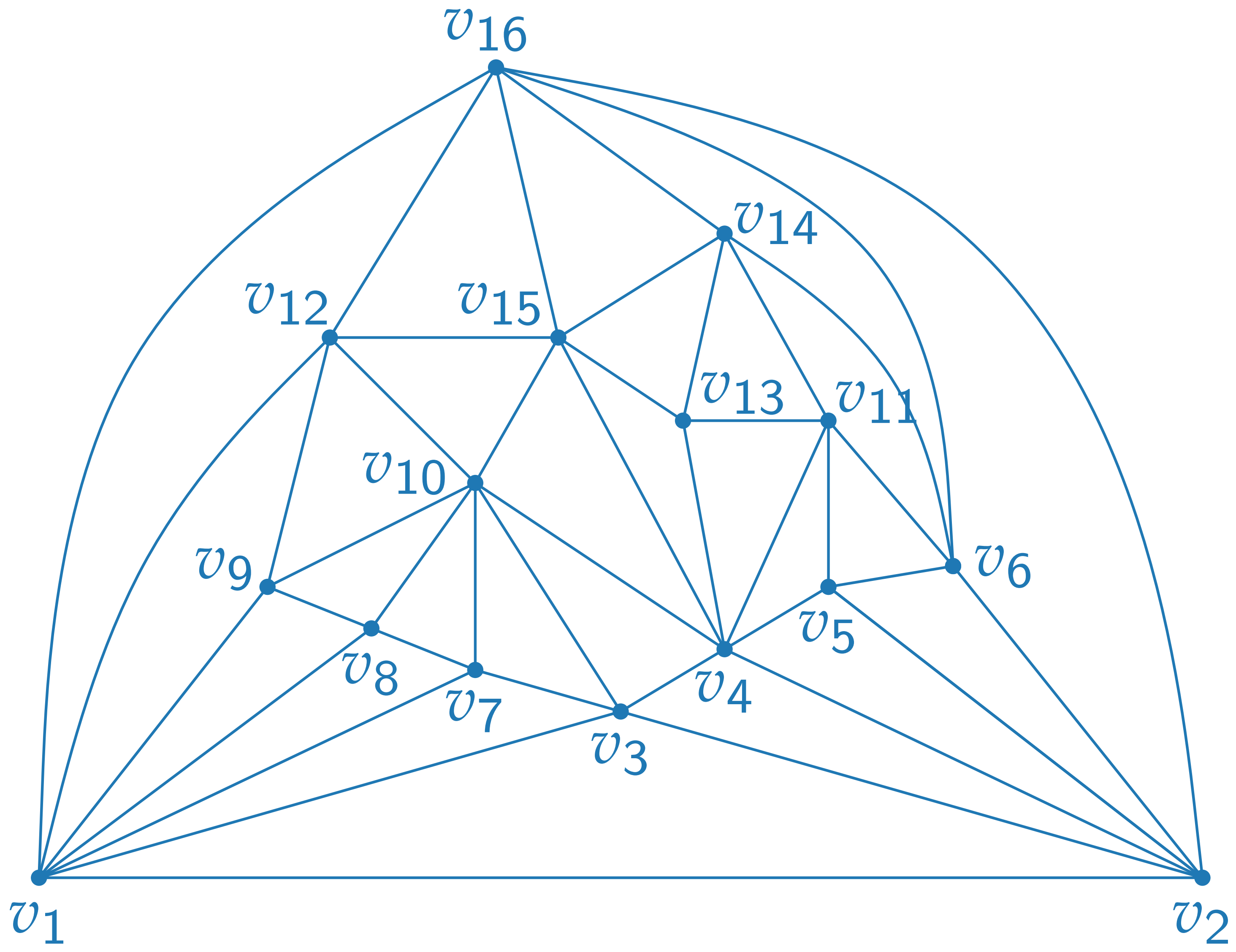
Canonical order – example



Canonical order – example



Canonical order – example



Canonical order – existence

Lemma.
Every triangulated plane graph has a canonical order.

Canonical order – existence

Lemma.
Every triangulated plane graph has a canonical order.

Proof.

- 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 order – existence

Lemma.
Every triangulated plane graph has a canonical order.

Proof.

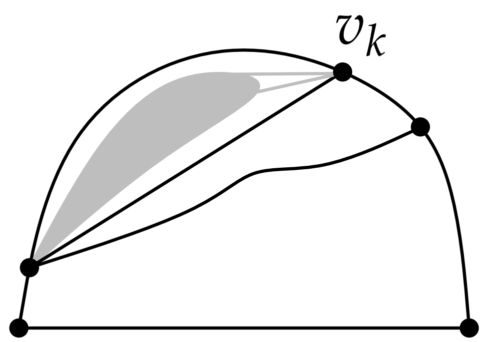
- Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions C1-C3 hold.
- Induction hypothesis: Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k+1 \le i \le n$.

Canonical order – existence

Lemma.
Every triangulated plane graph has a canonical order.

Proof.

- 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 $k+1 \leq i \leq n$.
- Induction step: Consider G_k. We search for v_k.

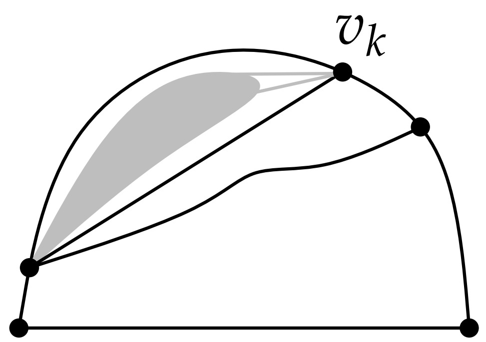


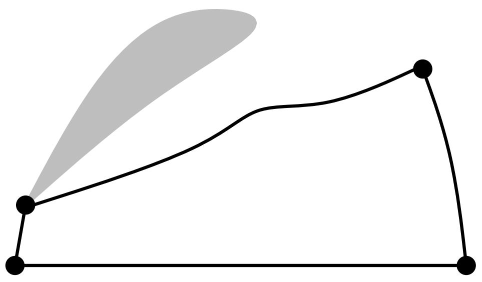
Canonical order – existence

Lemma.
Every triangulated plane graph has a canonical order.

Proof.

- 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 $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k+1 \leq i \leq n$.
- Induction step: Consider G_k. We search for v_k.





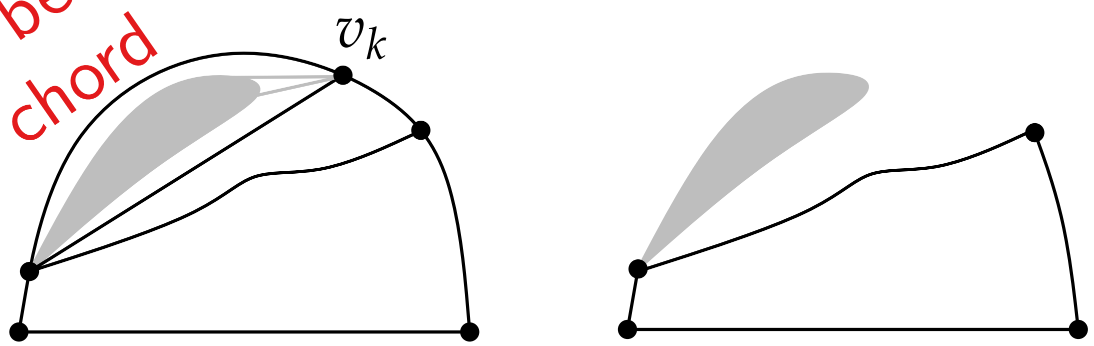
Canonical order – existence

Lemma.
Every triangulated plane graph has a canonical order.

Proof.

- 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 $k+1 \leq i \leq n$.
- Induction step: Consider G_k. We search for v_k.

v_k should not be adjacent to a chord



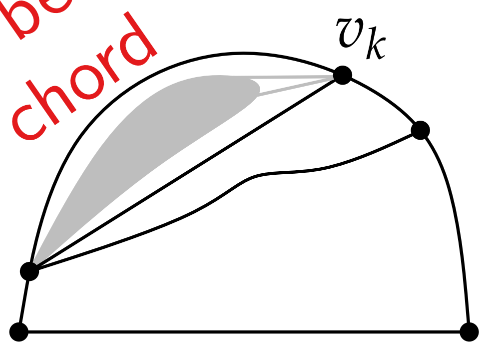
Canonical order – existence

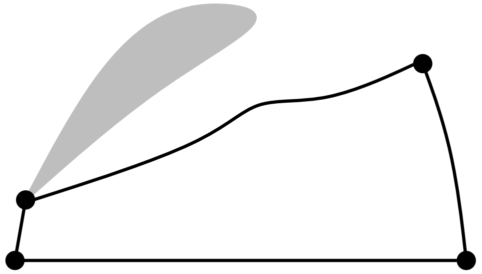
Lemma.
Every triangulated plane graph has a canonical order.

Proof.

- 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 $k+1 \leq i \leq n$.
- Induction step: Consider G_k. We search for v_k.

v_k should not be adjacent to a chord





Have to show:

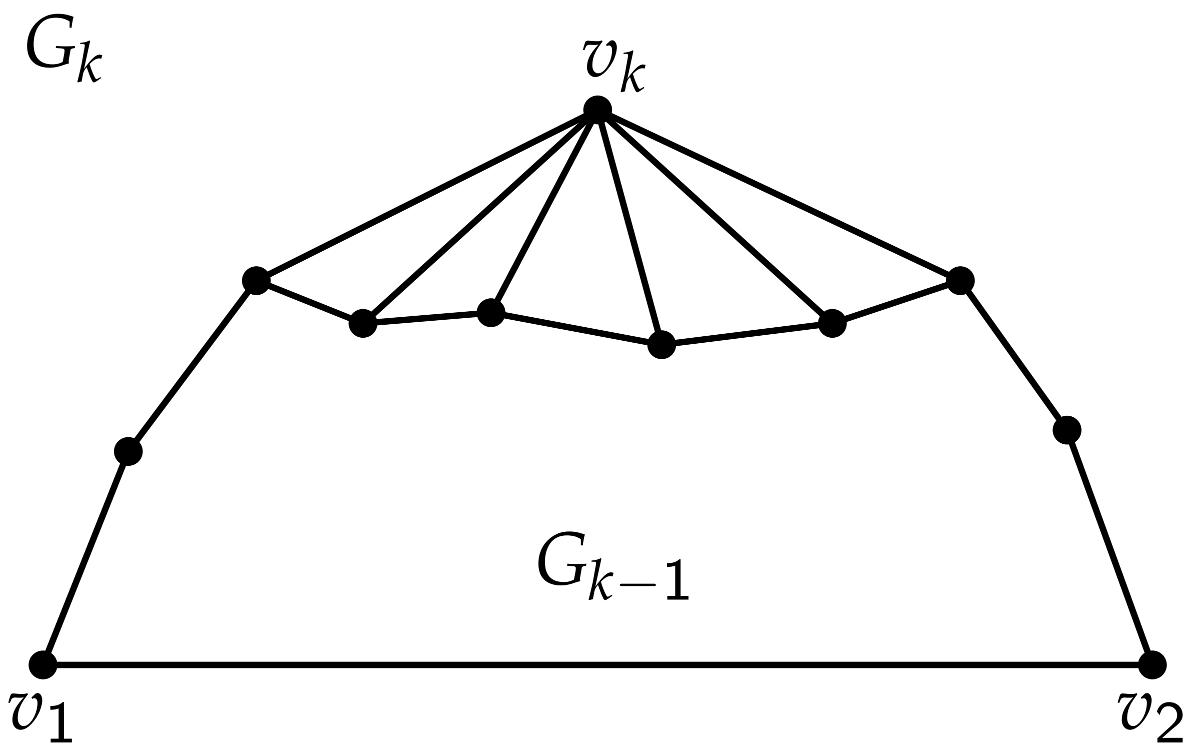
1. v_k not adjacent to chord is sufficient
2. Such v_k exists

Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.

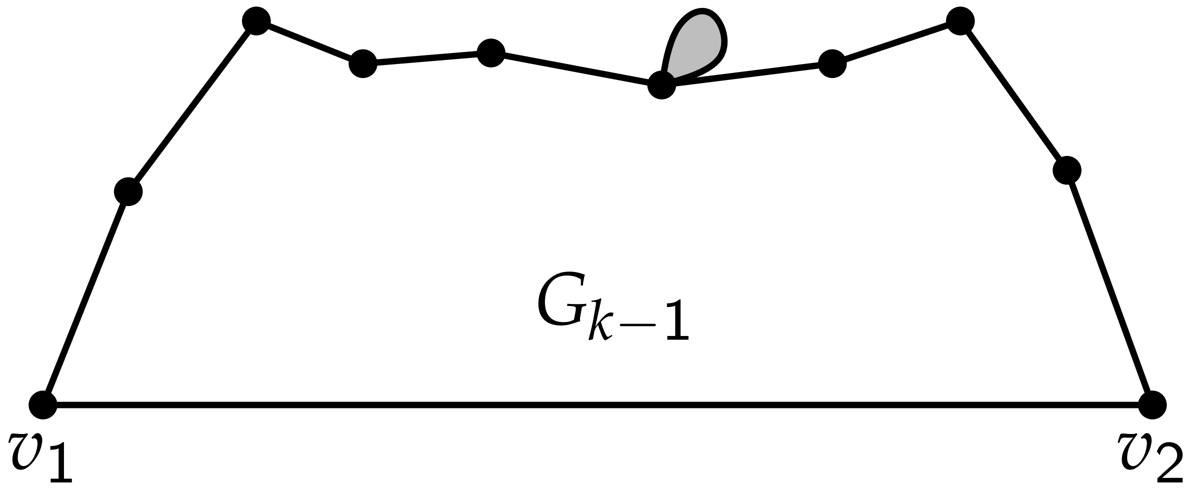
Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



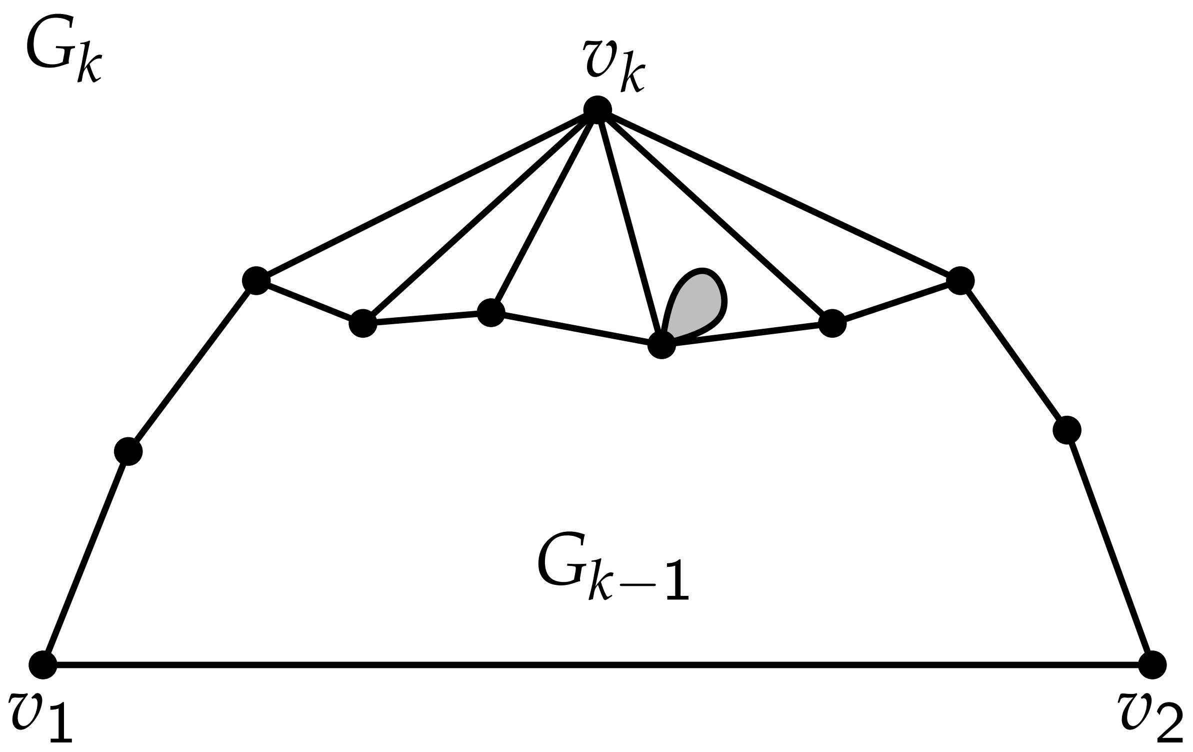
Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



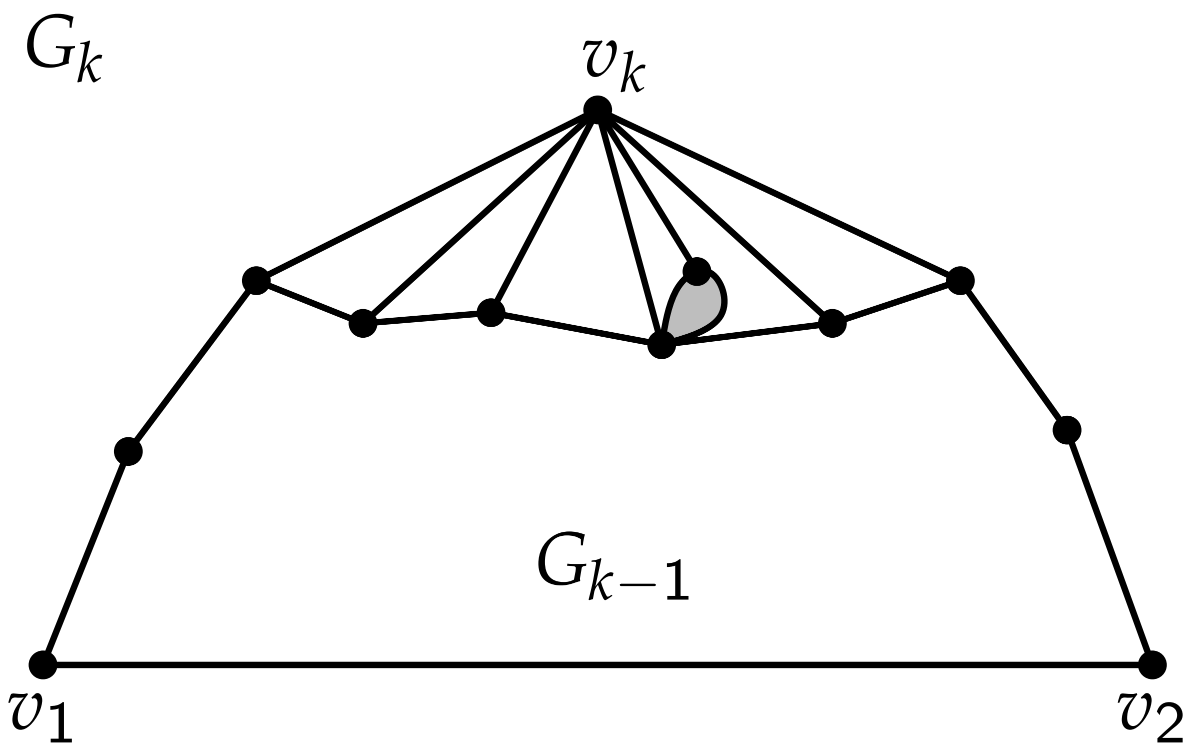
Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



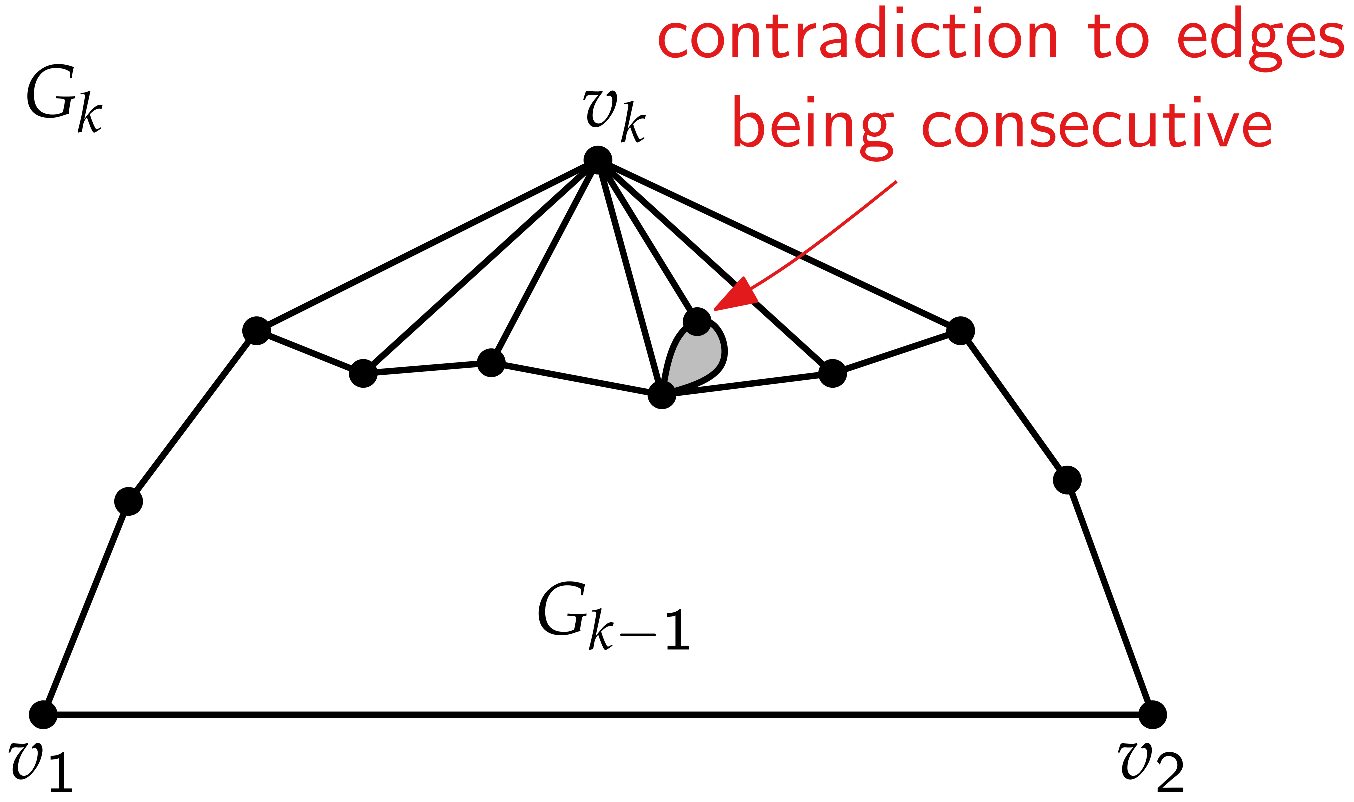
Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



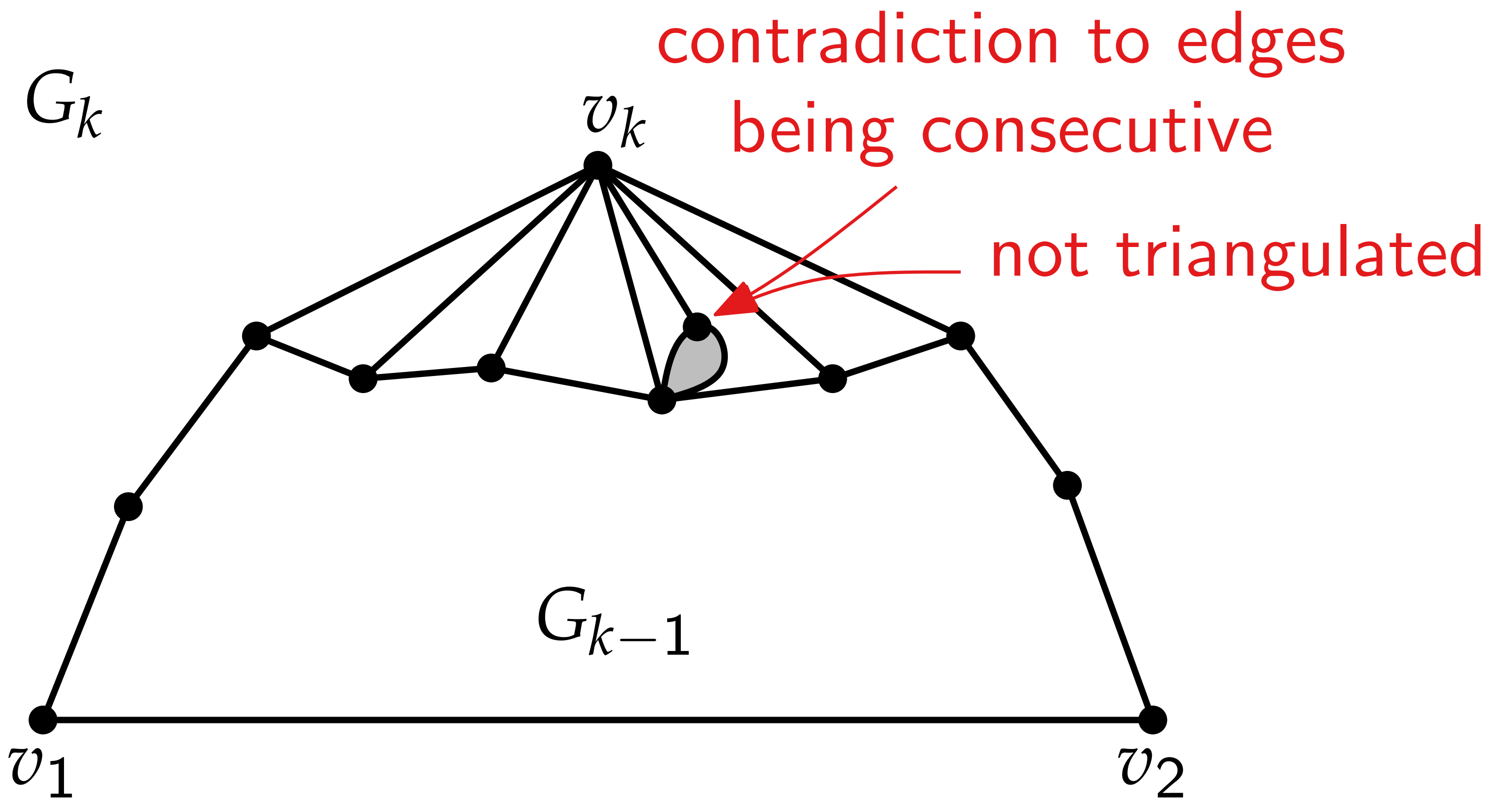
Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



Canonical order – existence

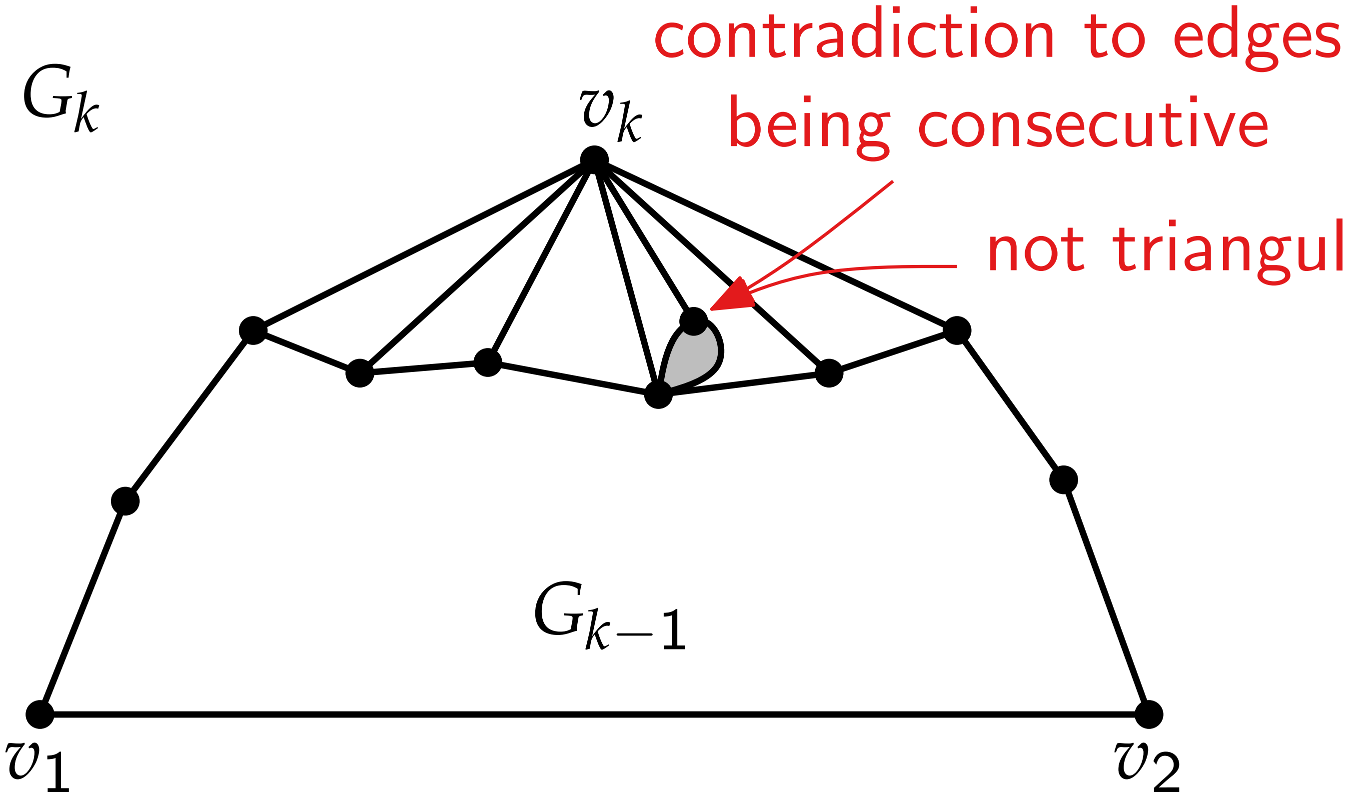
Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.

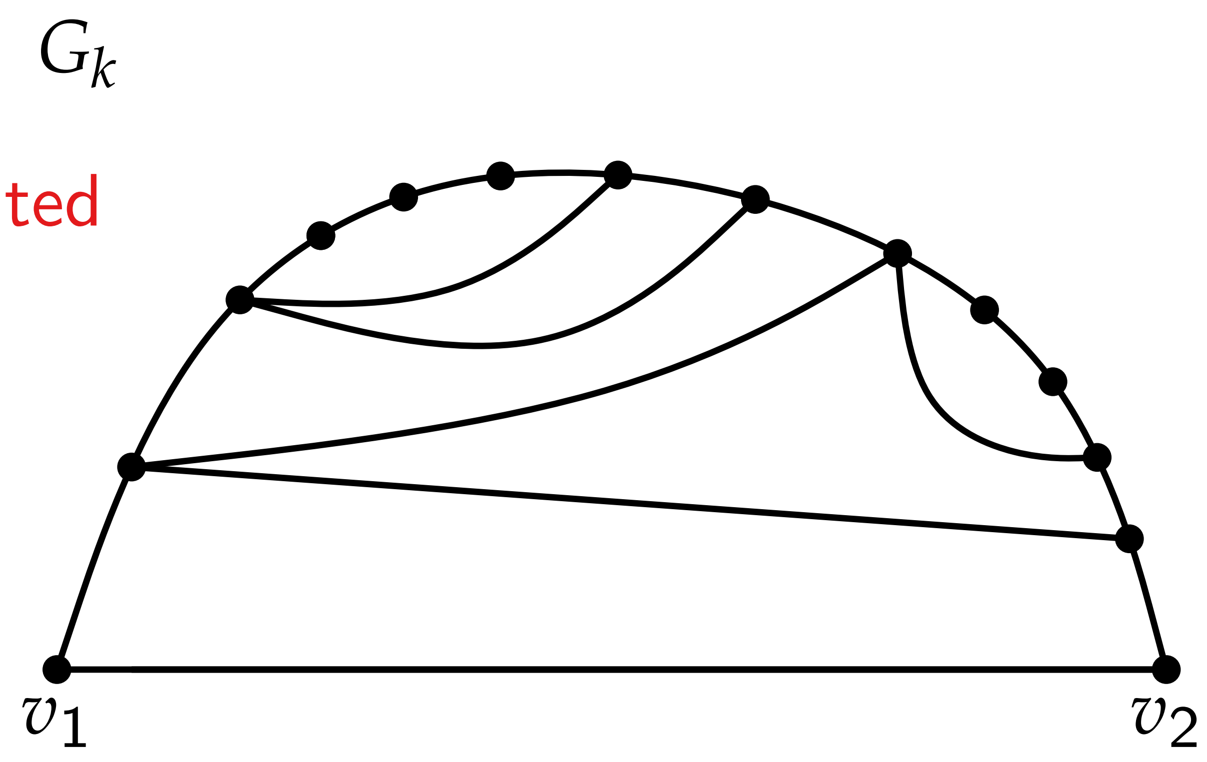


Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.

Claim 2. There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.

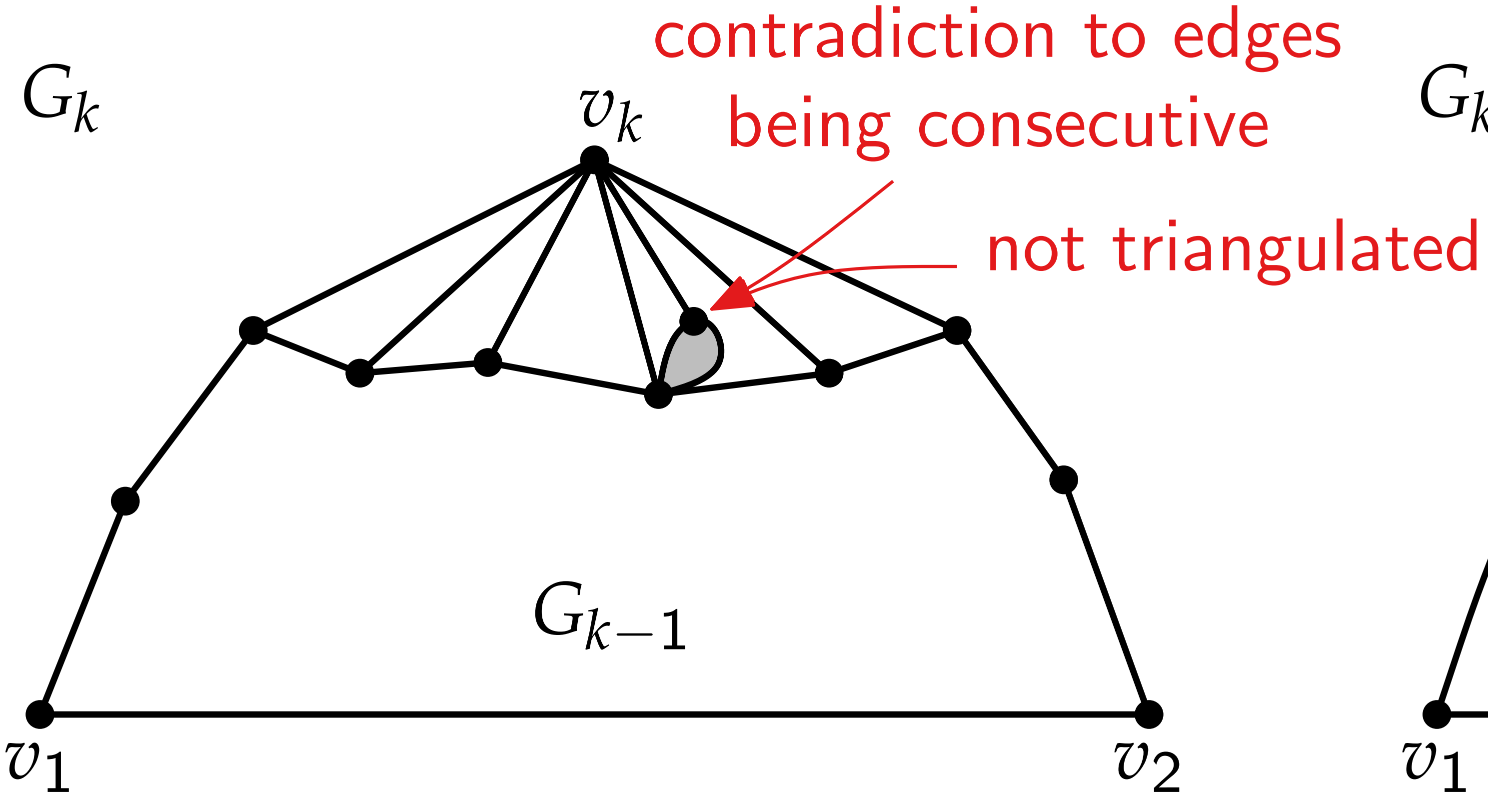


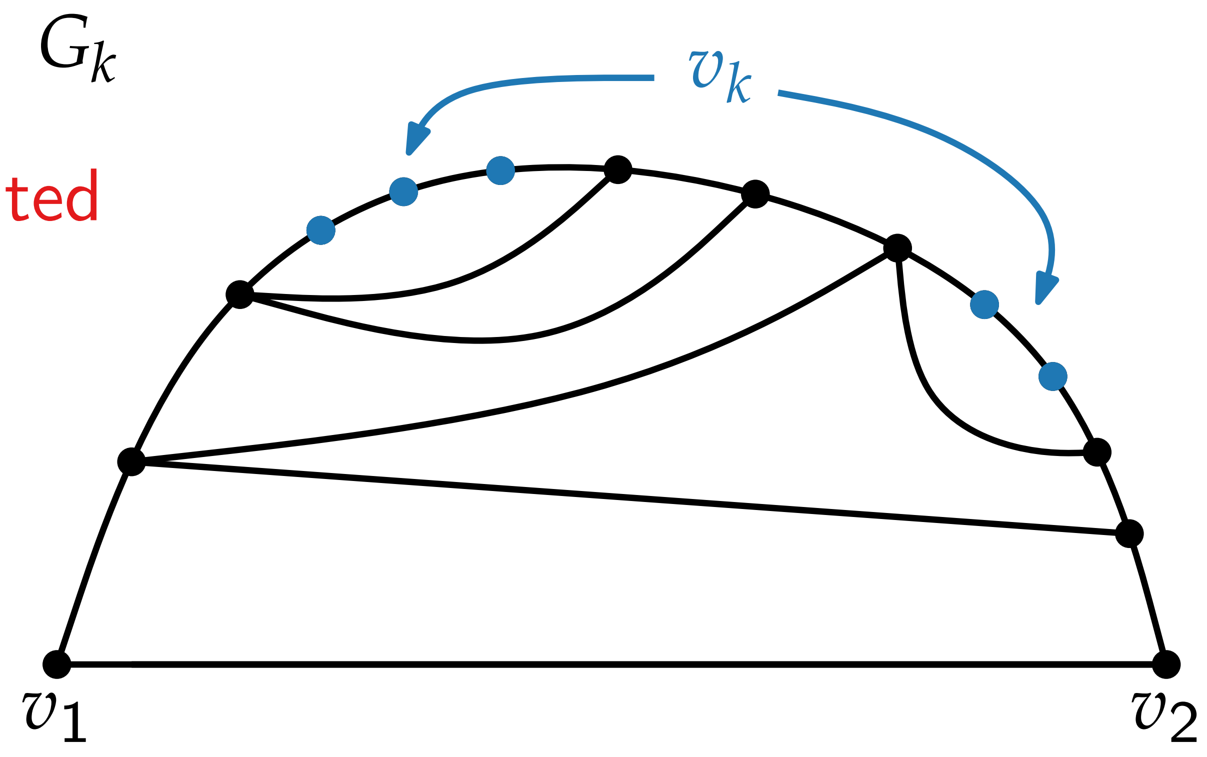


Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.

Claim 2. There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.

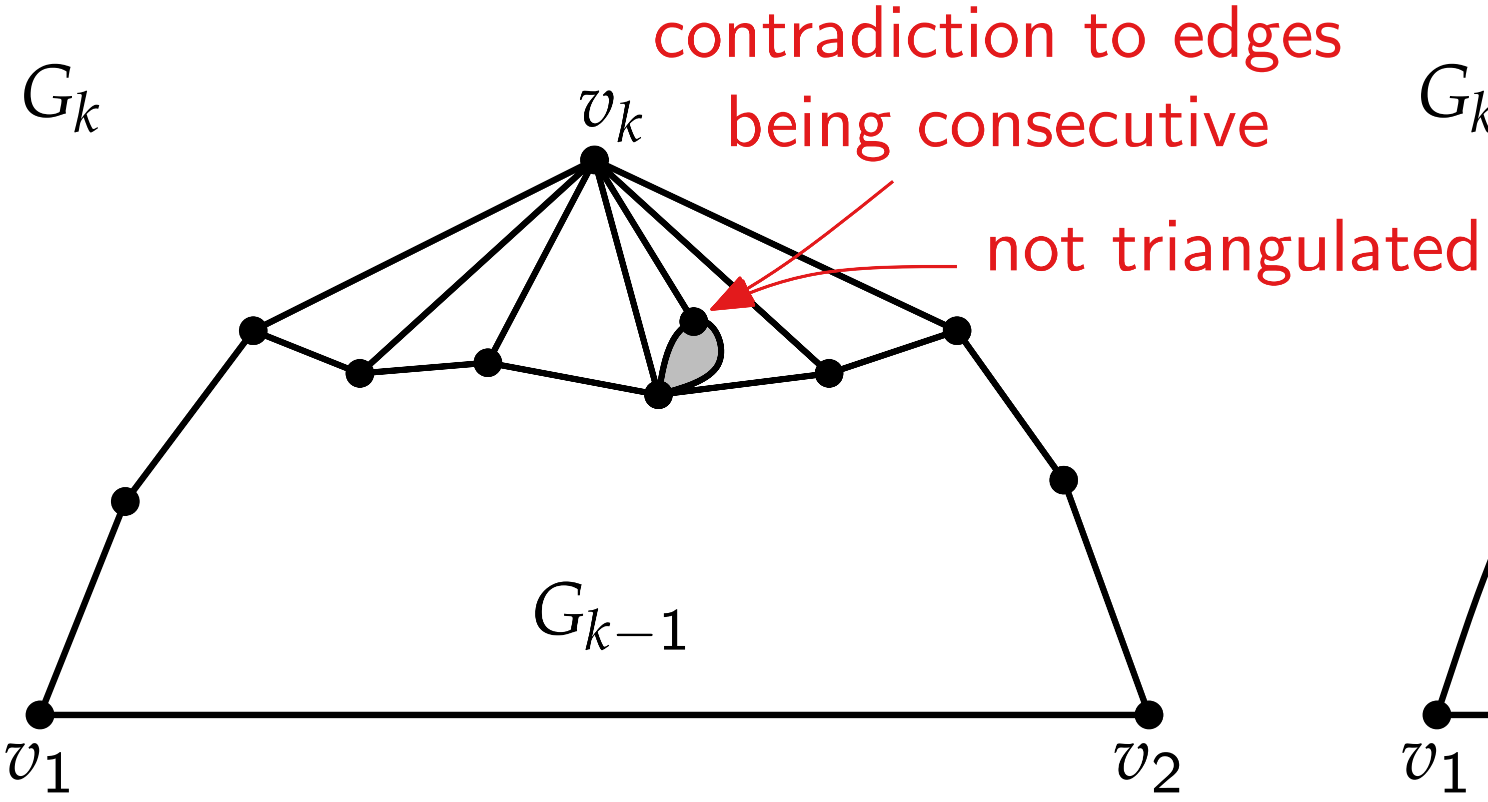


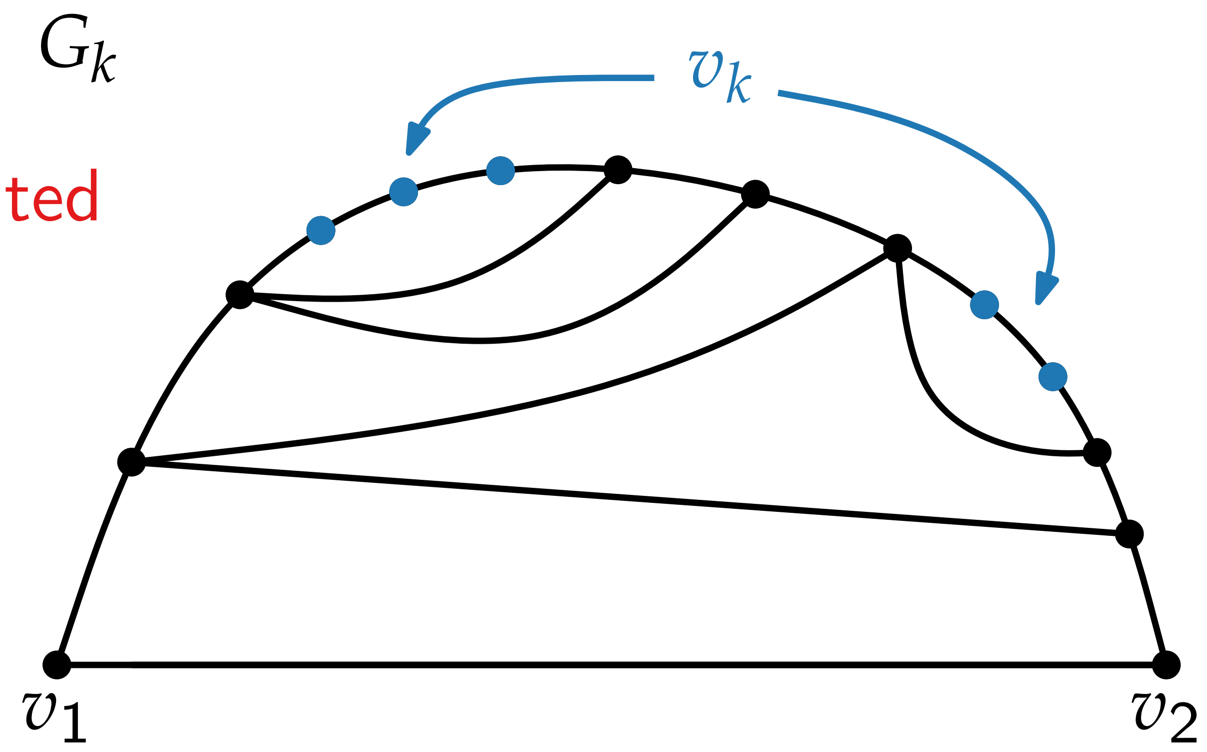


Canonical order – existence

Claim 1. If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.

Claim 2.
There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.





This completes proof of Lemma. □

Canonical order – implementation

Algorithm CanonicalOrder

forall $v \in V$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false;

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **to** 3 **do**
  choose $v \neq v_1, v_2$ such that mark(v) $=$ false,
   out(v) $=$ true, and chords(v) $= 0$
  $v_k \leftarrow v$; mark(v) $\leftarrow$ 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 of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$
  update number of chords for w_i and its neighbours

- chord(v) – # chords adjacent to v
- mark(v) $=$ true iff vertex v was numbered
- out(v) $=$ true iff v is currently outer vertex

Canonical order – implementation

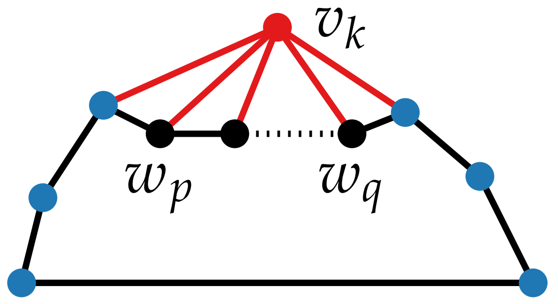
Algorithm CanonicalOrder

forall $v \in V$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false;

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **to** 3 **do**
  choose $v \neq v_1, v_2$ such that mark(v) $=$ false, out(v) $=$ true, and chords(v) $= 0$
  $v_k \leftarrow v$; mark(v) $\leftarrow$ 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 of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$
  update number of chords for w_i and its neighbours

- chord(v) – # chords adjacent to v
- mark(v) $=$ true iff vertex v was numbered
- out(v) $=$ true iff v is currently outer vertex



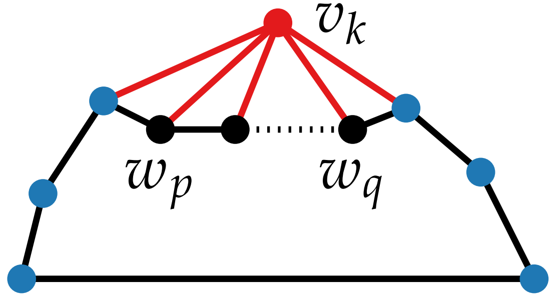
Canonical order – implementation

Algorithm CanonicalOrder

forall $v \in V$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false;

out(v_1), out(v_2), out(v_n) $\leftarrow$ true
for $k = n$ **to** 3 **do**
  choose $v \neq v_1, v_2$ such that mark(v) $=$ false, out(v) $=$ true, and chords(v) $= 0$
  $v_k \leftarrow v$; mark(v) $\leftarrow$ 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 of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$
  update number of chords for w_i and its neighbours

- chord(v) – # chords adjacent to v
- mark(v) $=$ true iff vertex v was numbered
- out(v) $=$ true iff v is currently outer vertex



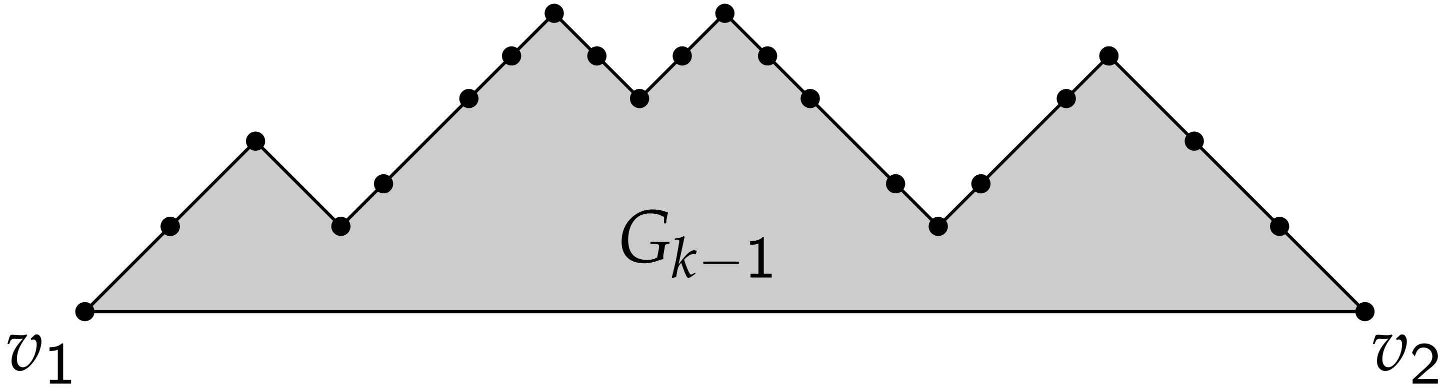
Lemma.
Algorithm CanonicalOrder computes a canonical order of a plane graph in $\mathcal{O}(n)$ time.

Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-4,0)$,
- 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 ± 1.

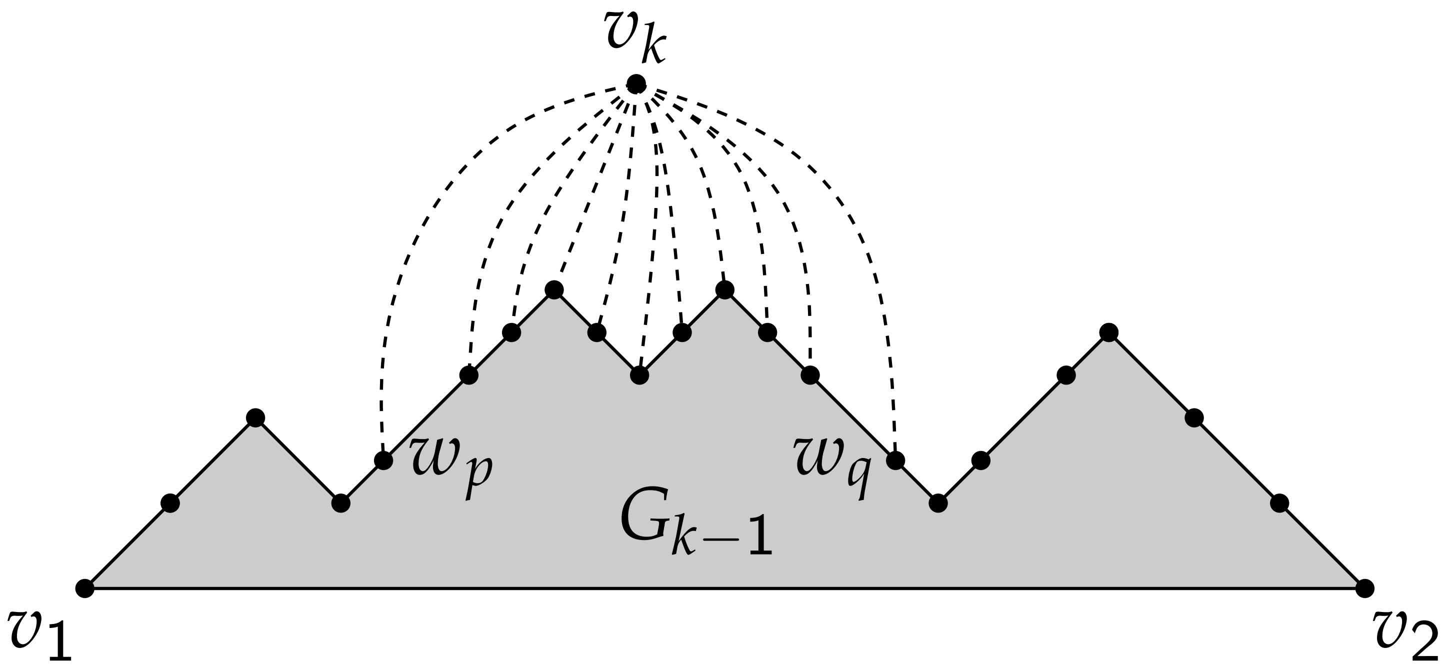


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-4,0)$,
- 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 ± 1.

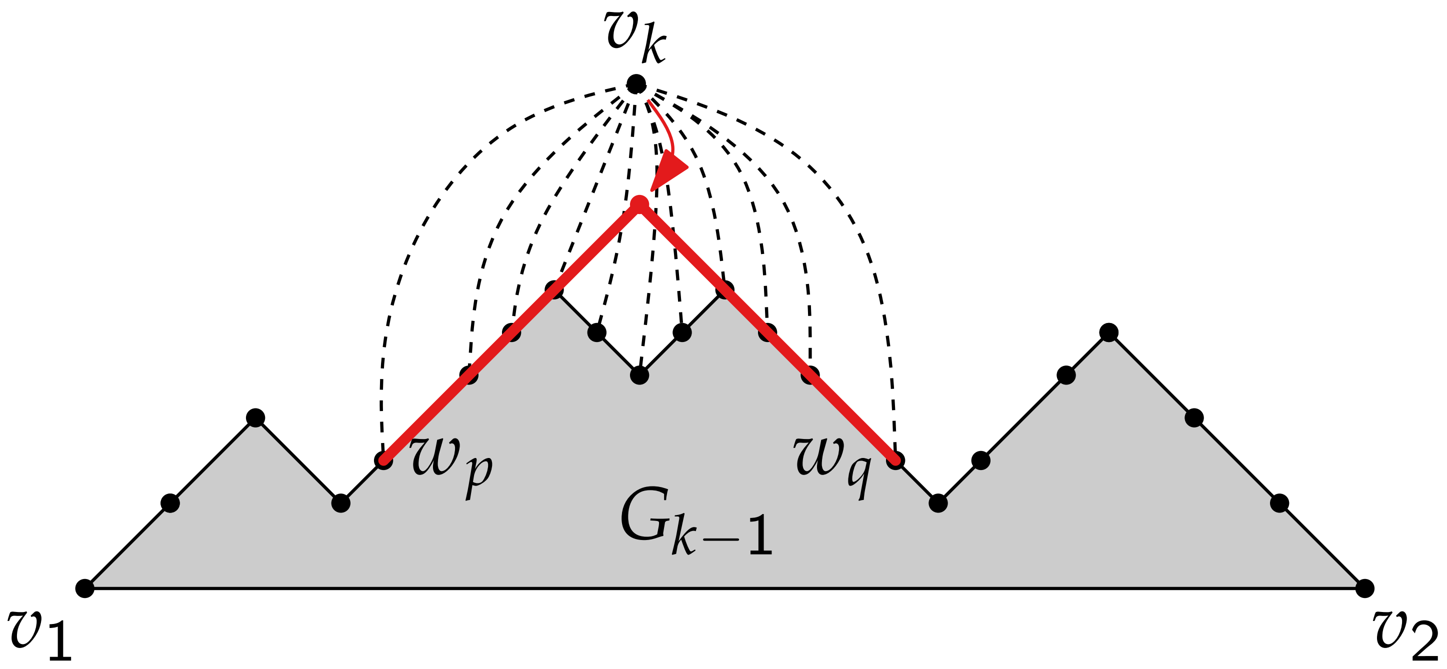


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0, 0)$, v_2 is on $(2k-4, 0)$,
- 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 ± 1.

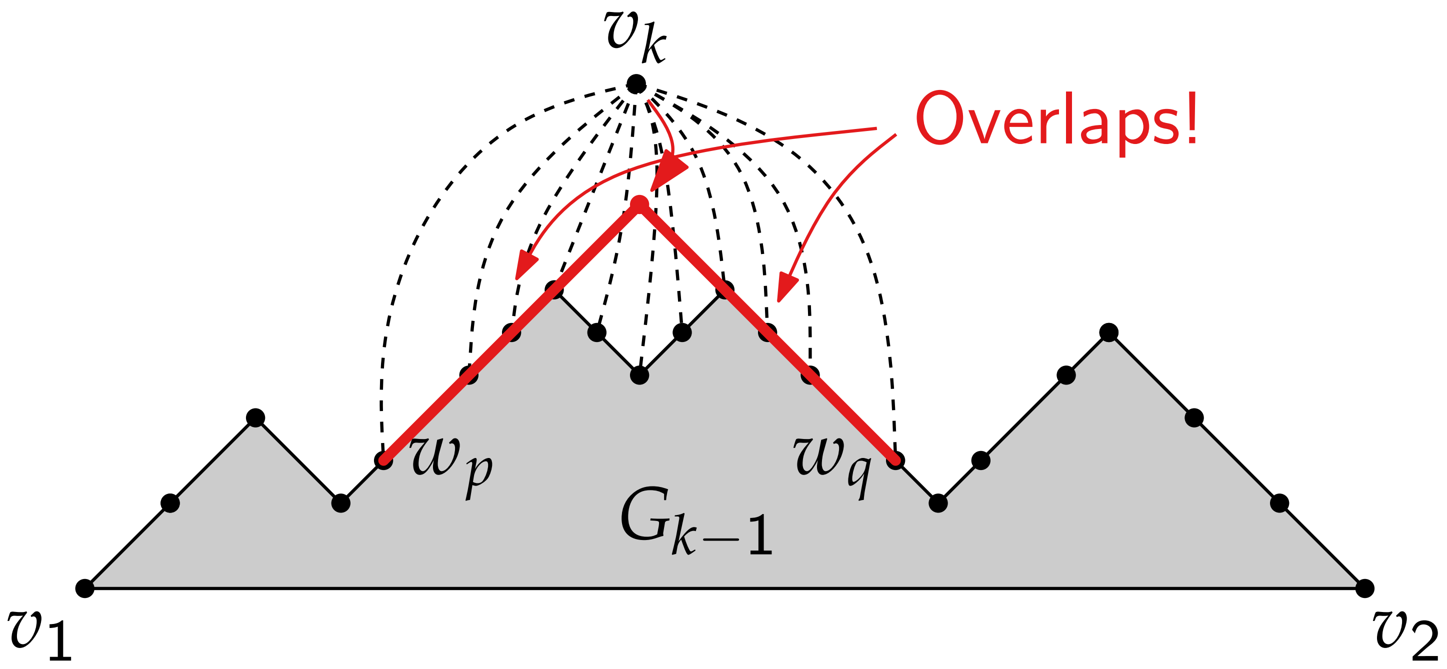


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0, 0)$, v_2 is on $(2k-4, 0)$,
- 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 ± 1.

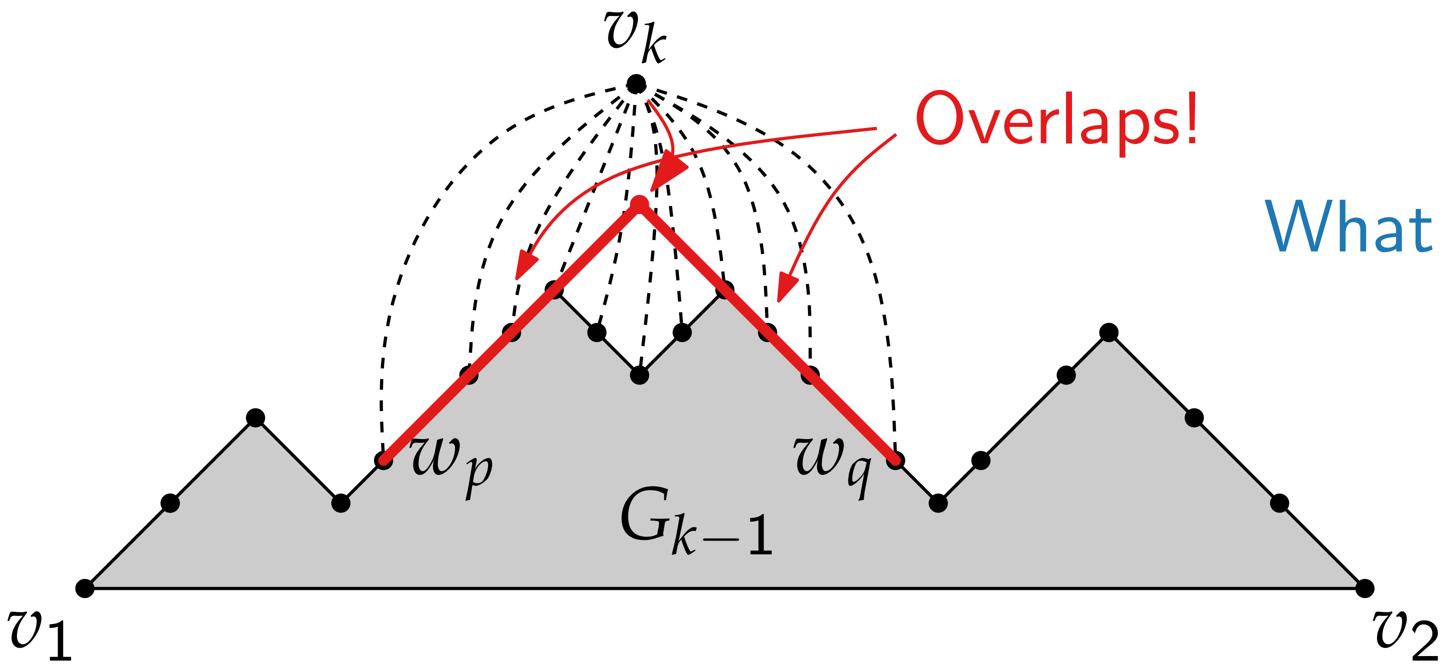


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-4,0)$,
- 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 ± 1.



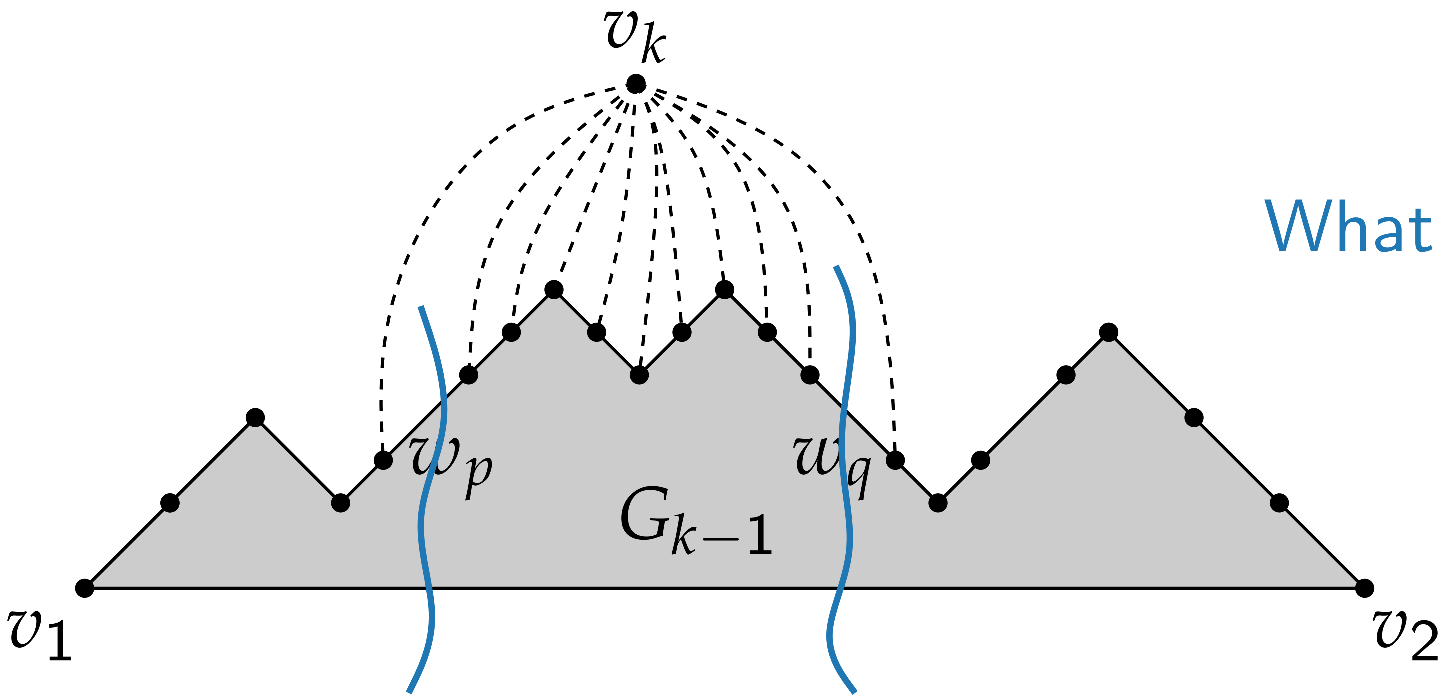
What could be the solution?

Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-4,0)$,
- 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 ± 1.



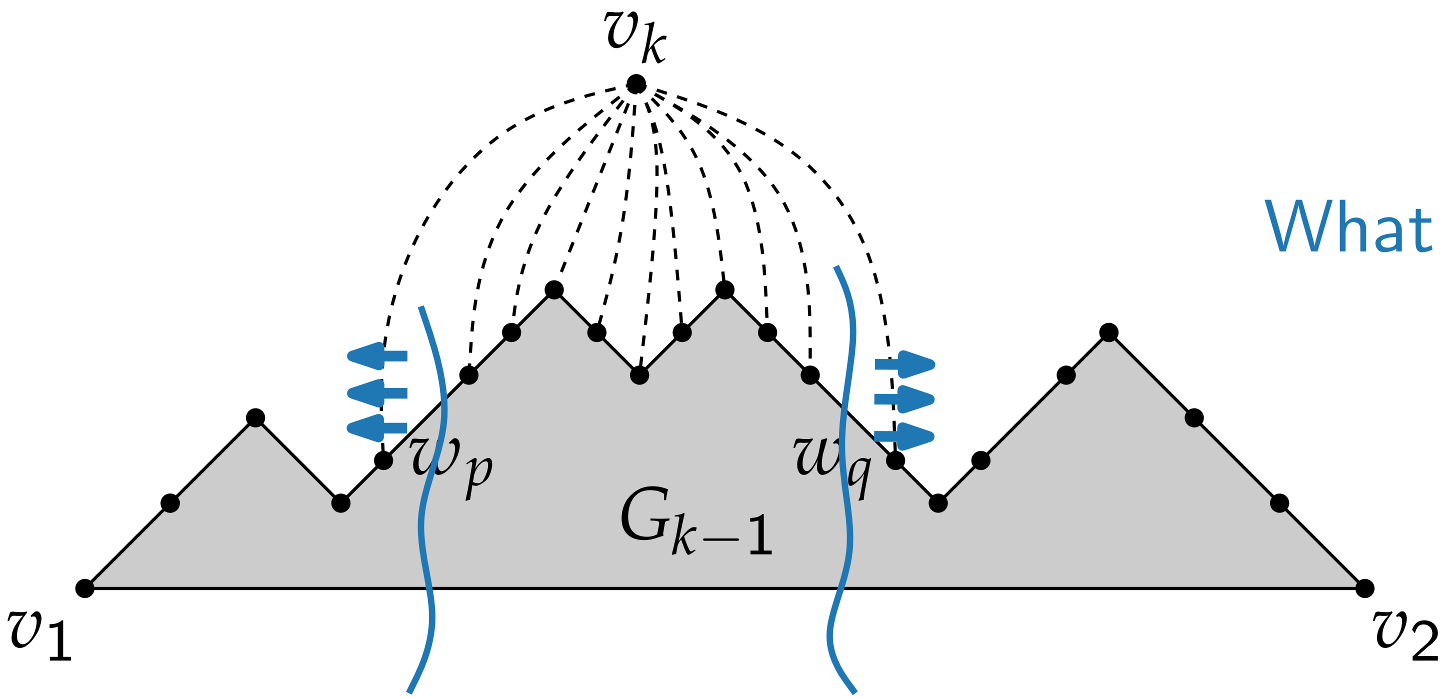
What could be the solution?

Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0, 0)$, v_2 is on $(2k-4, 0)$,
- 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 ± 1.



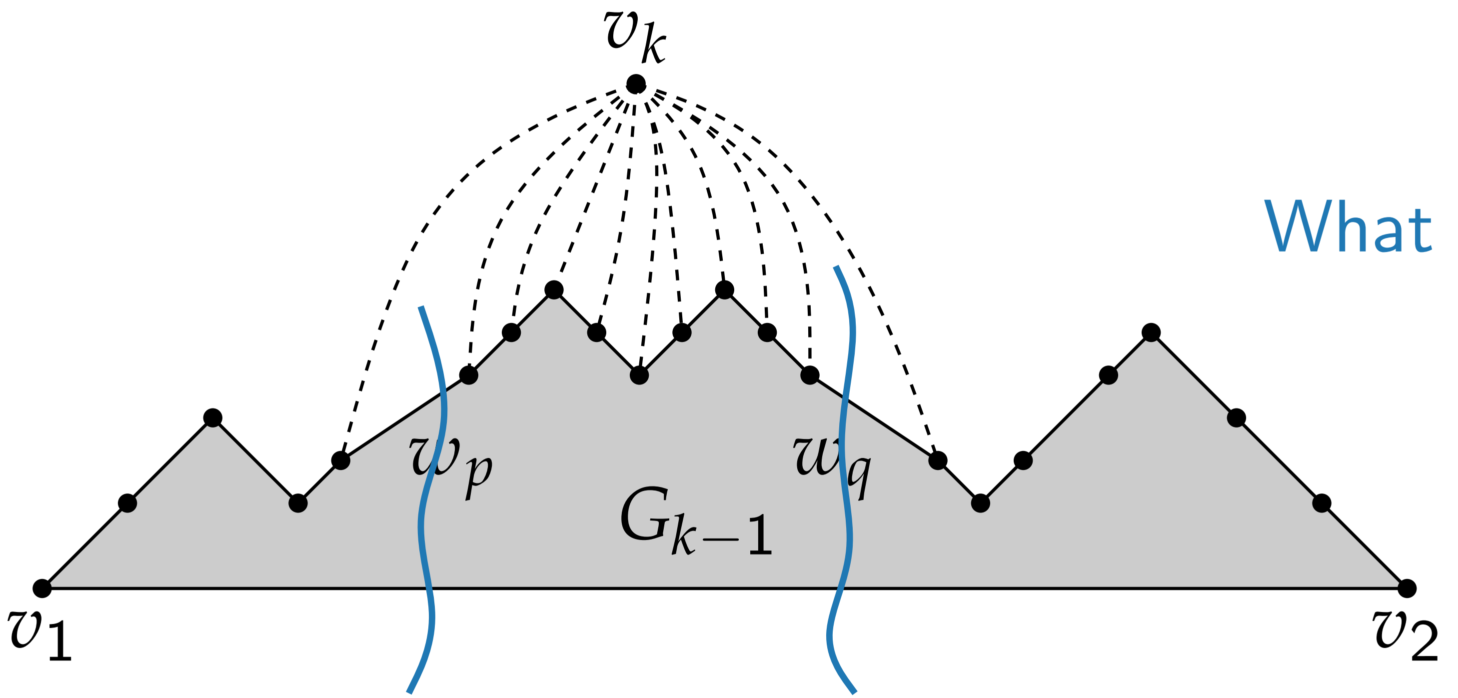
What could be the solution?

Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0, 0)$, v_2 is on $(2k-4, 0)$,
- 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 ± 1.



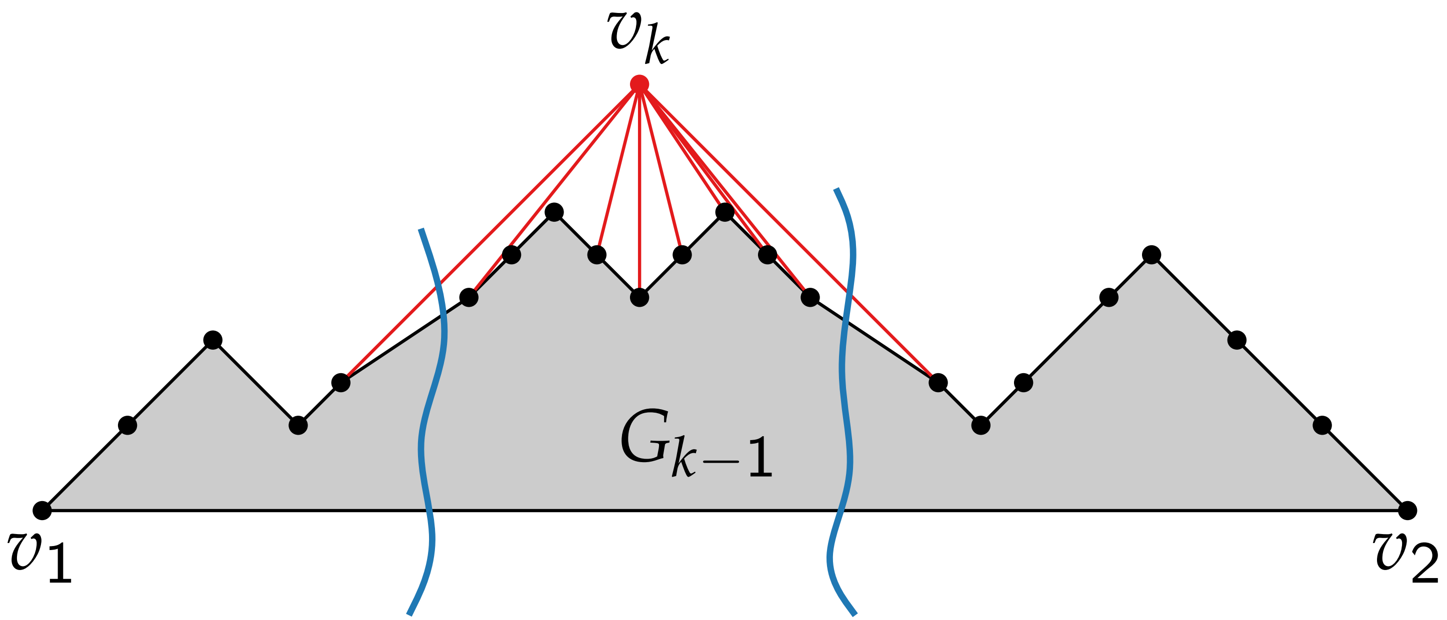
What could be the solution?

Shift method

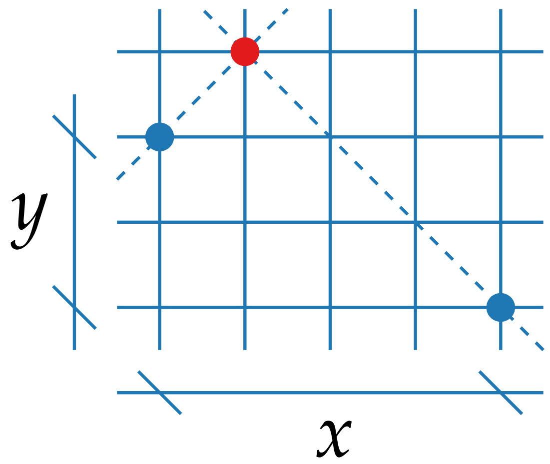
Algorithm invariants/constraints:

G_{k-1} is drawn such that

- v_1 is on $(0, 0)$, v_2 is on $(2k - 4, 0)$,
- 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 ± 1.



Shift method

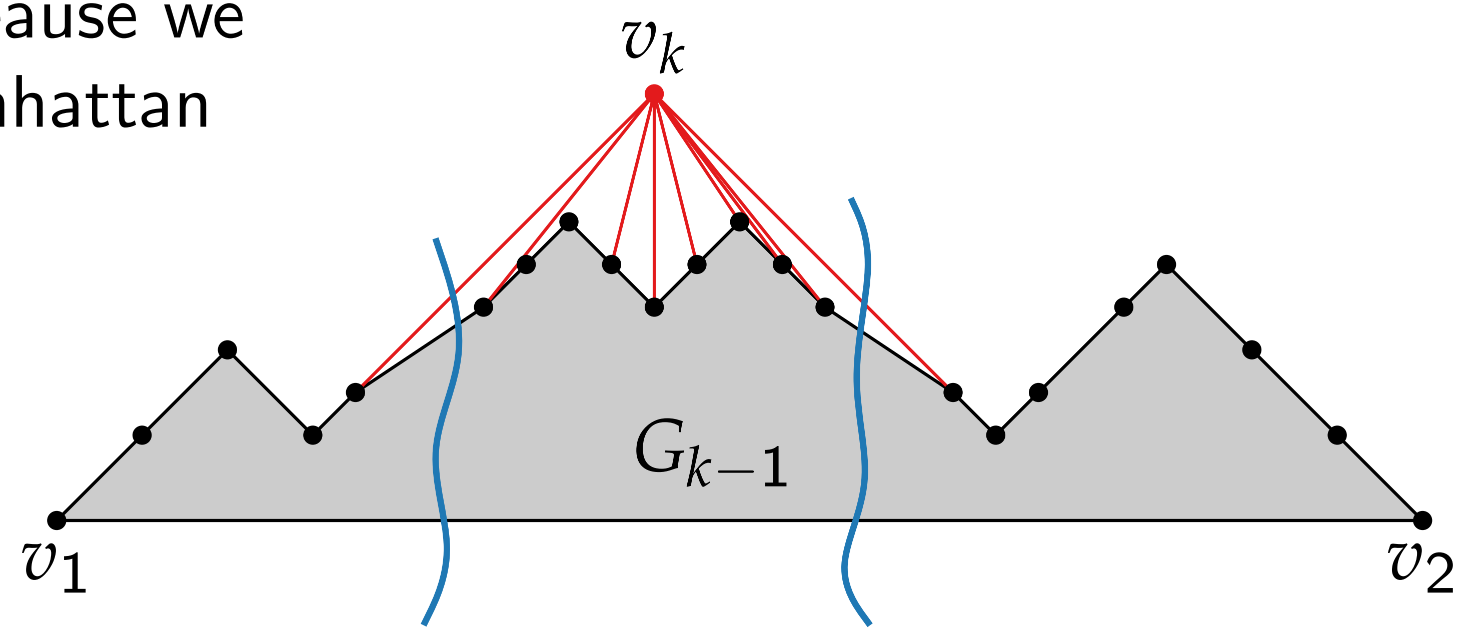


Algorithm invariants/constraints:

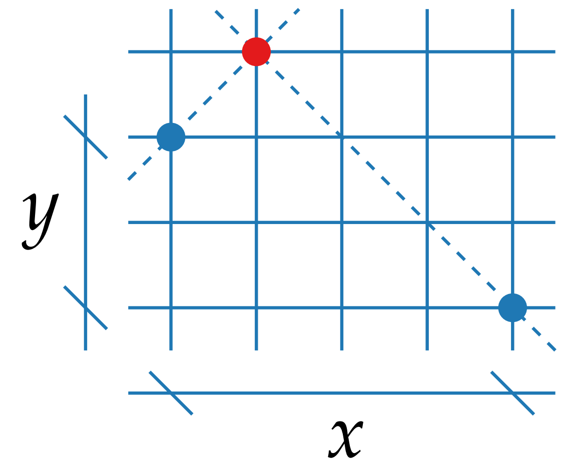
G_{k-1} is drawn such that

- v_1 is on $(0, 0)$, v_2 is on $(2k - 4, 0)$,
- 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 ± 1.

- v_k on grid, beause we had even Manhattan distance



Shift method

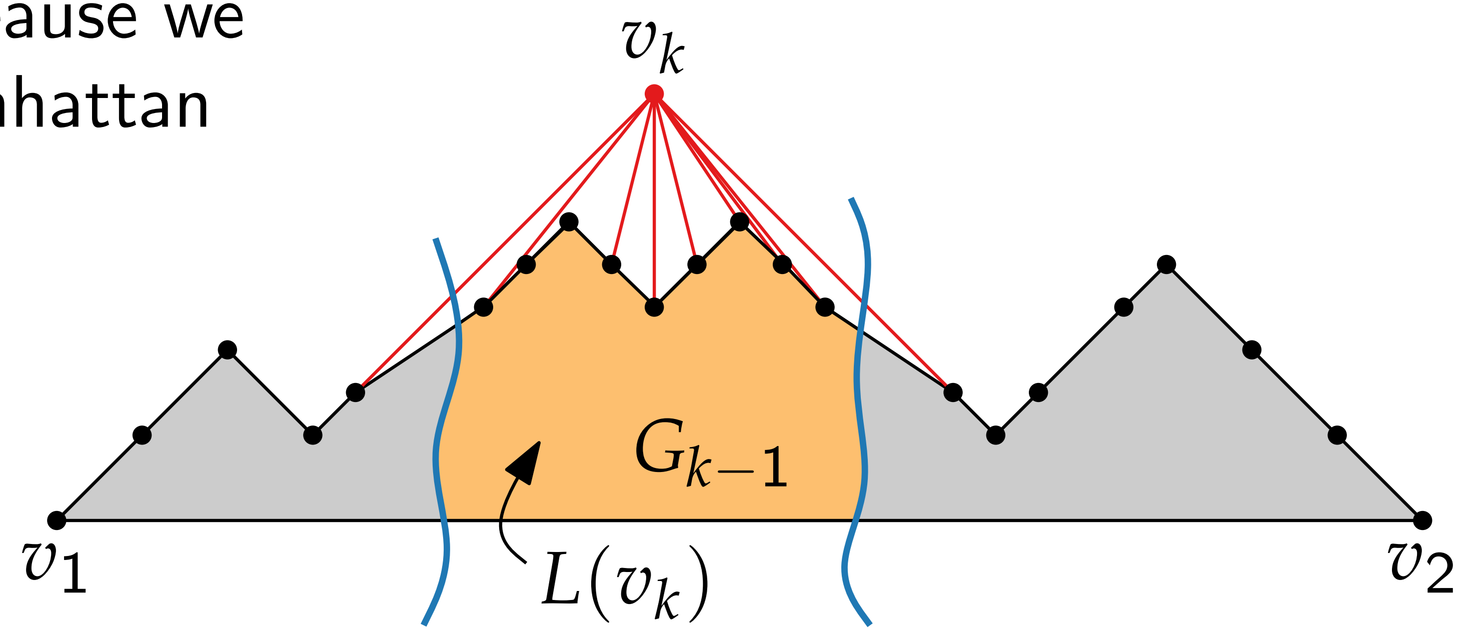


- v_k on grid, beause we had even Manhattan distance

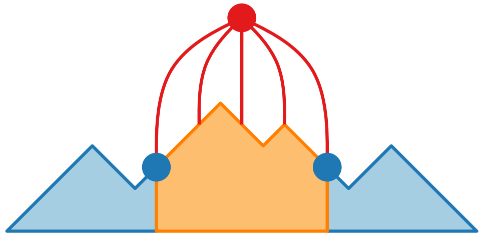
Algorithm invariants/constraints:

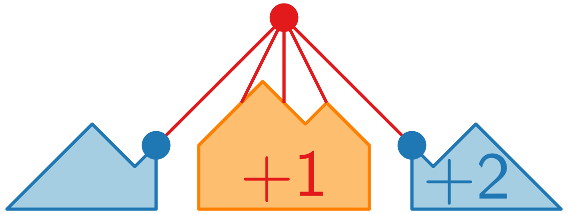
G_{k-1} is drawn such that

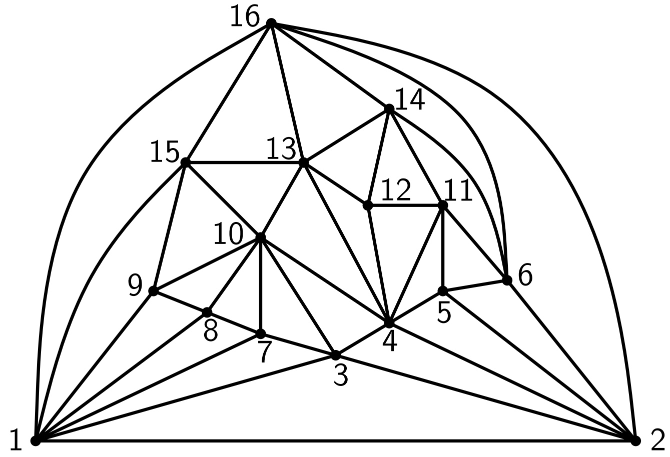
- v_1 is on $(0, 0)$, v_2 is on $(2k-4, 0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.



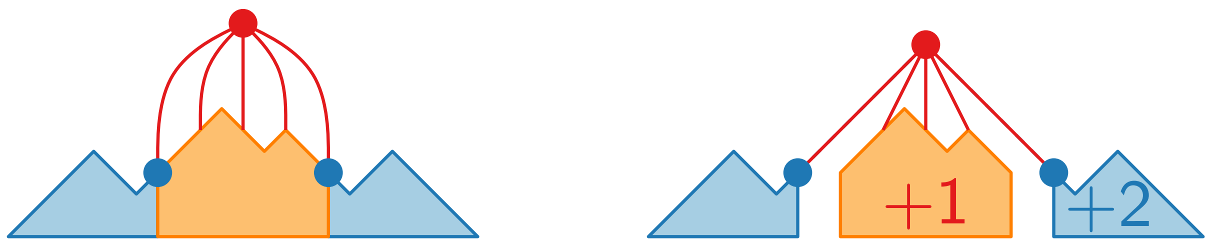
Shift method – example

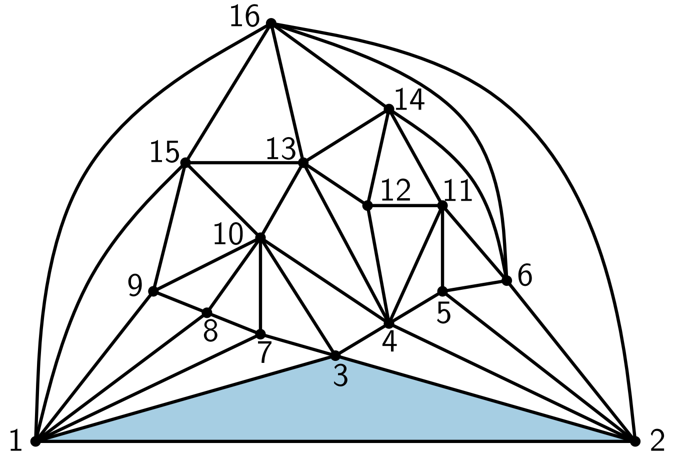


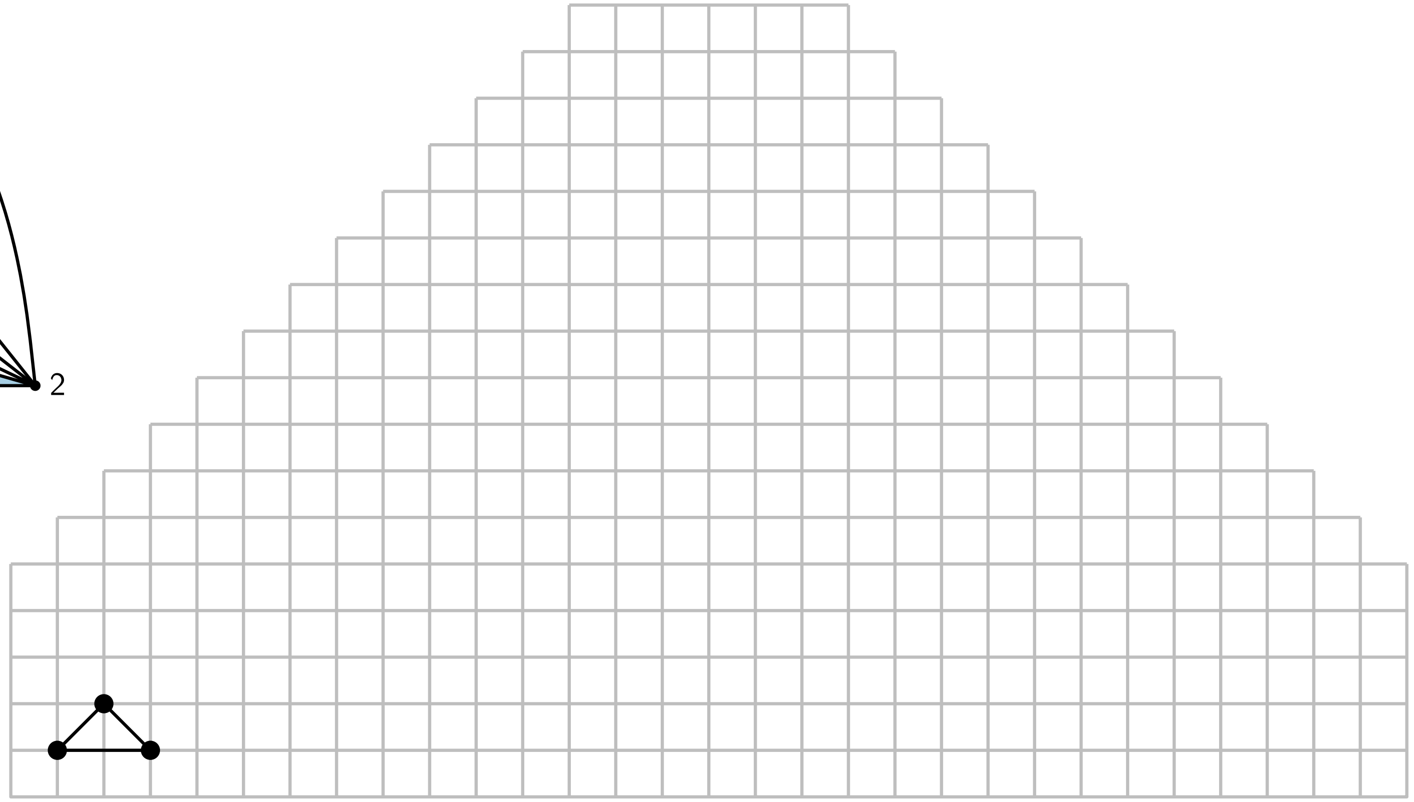




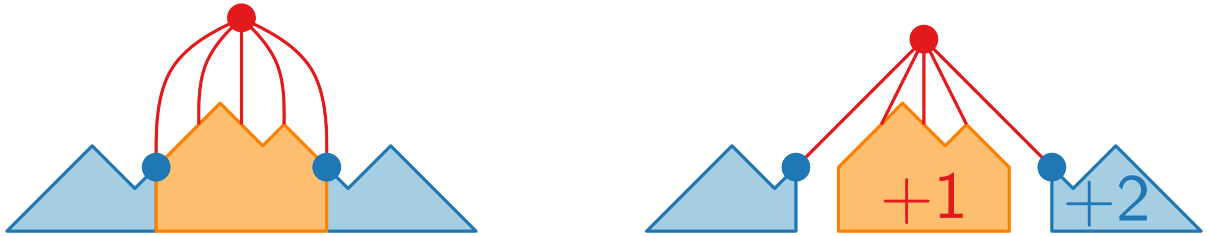
Shift method – example

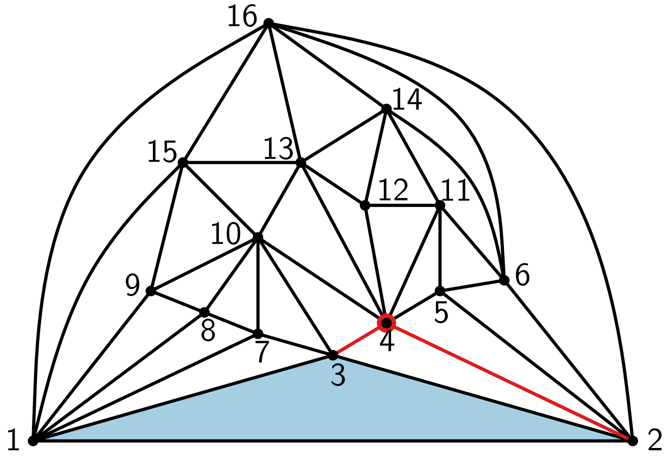


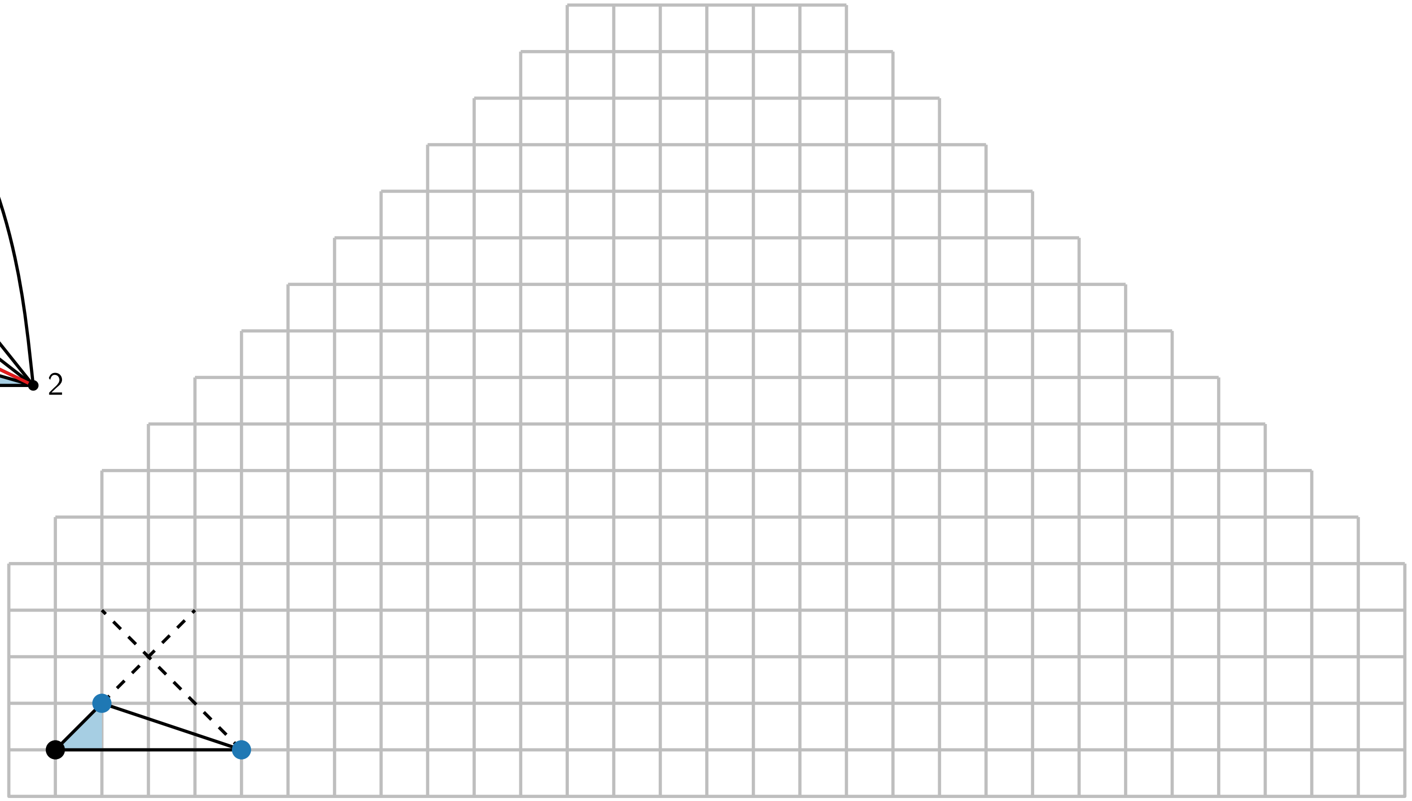




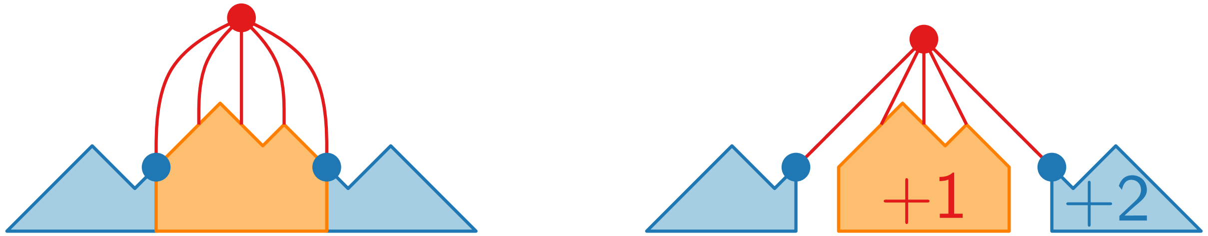
Shift method – example

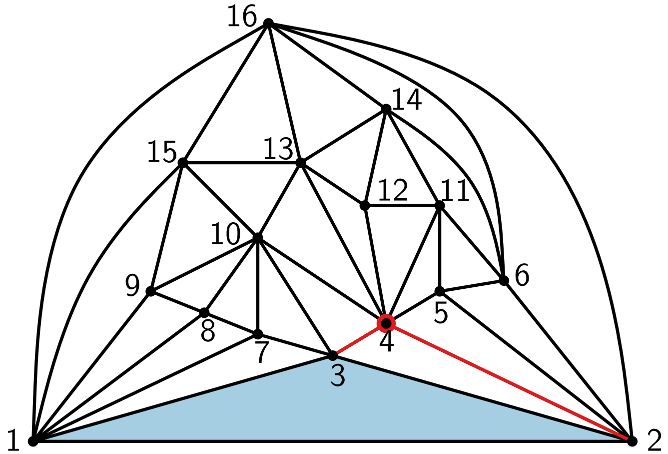


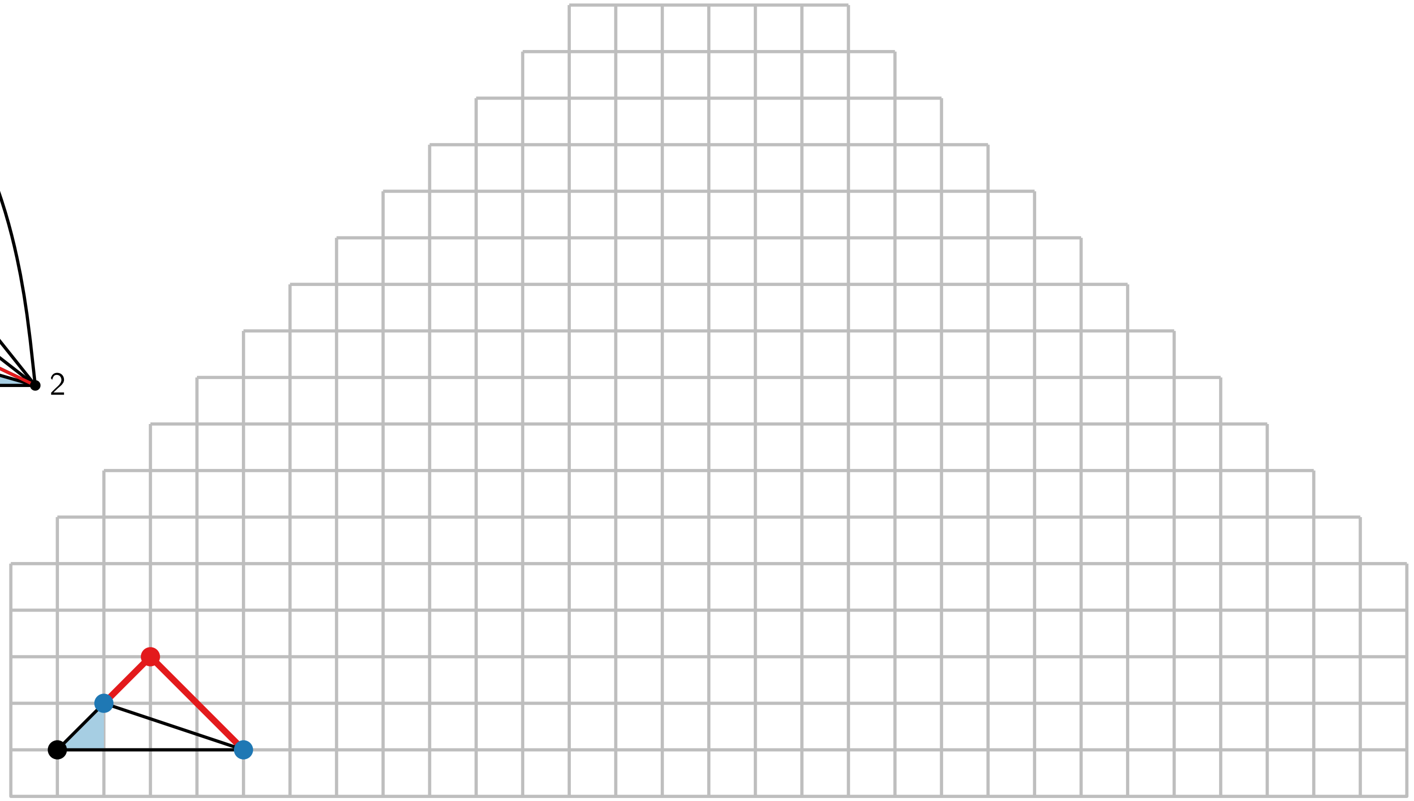




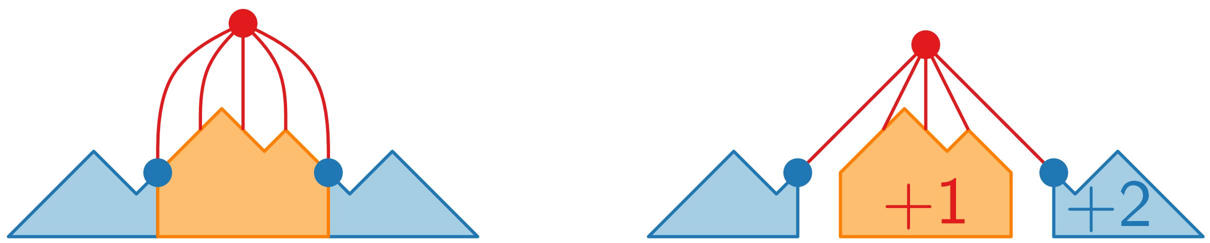
Shift method – example

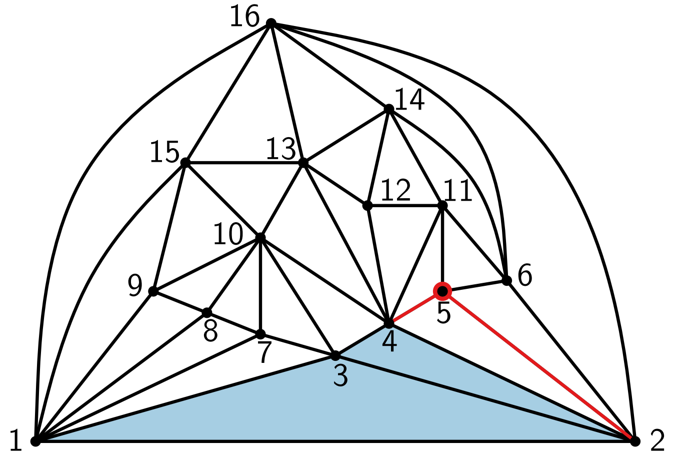


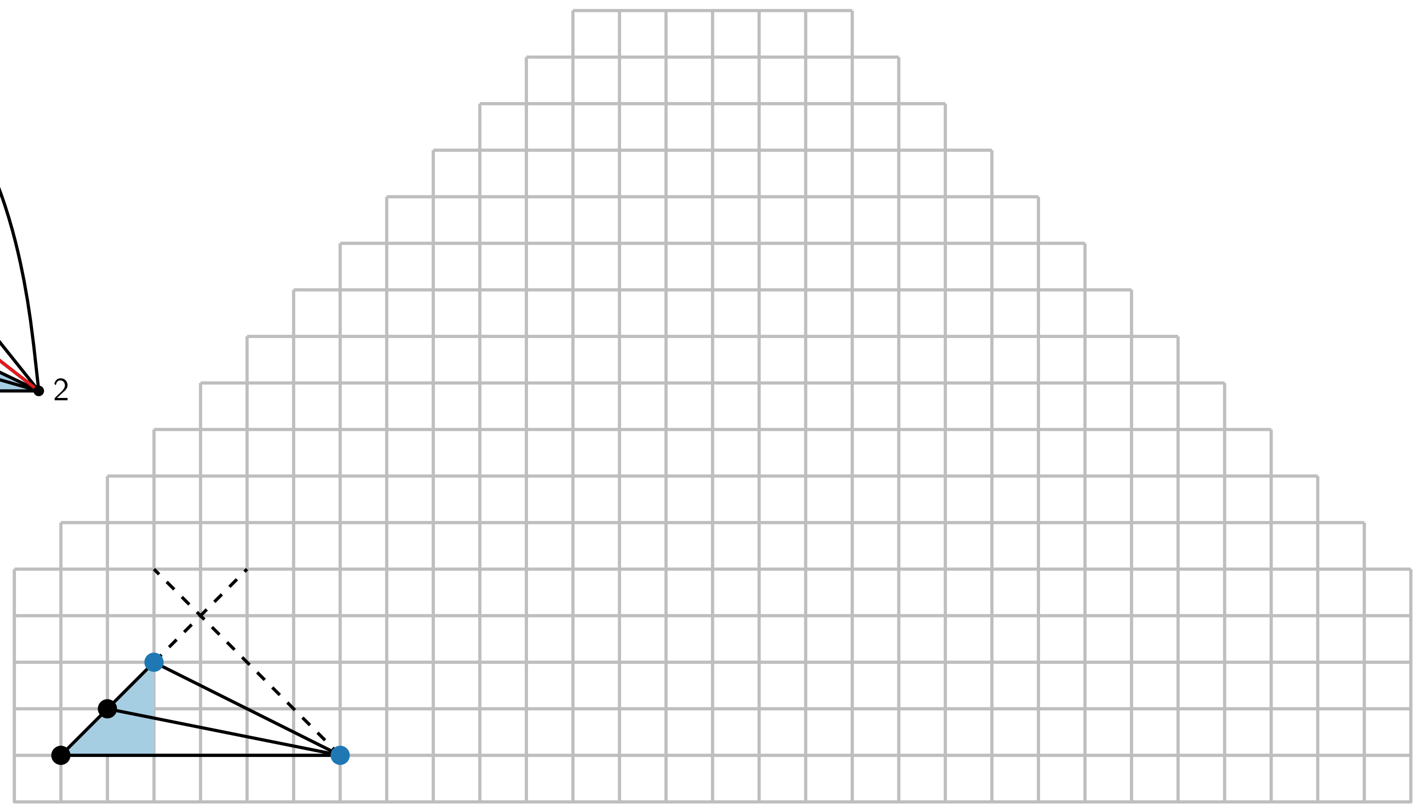




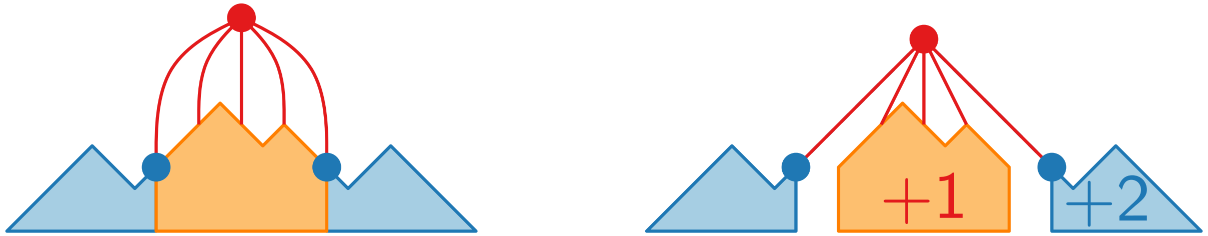
Shift method – example

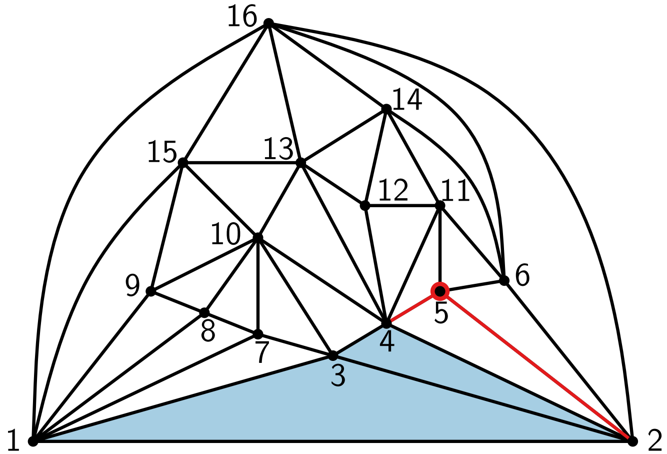


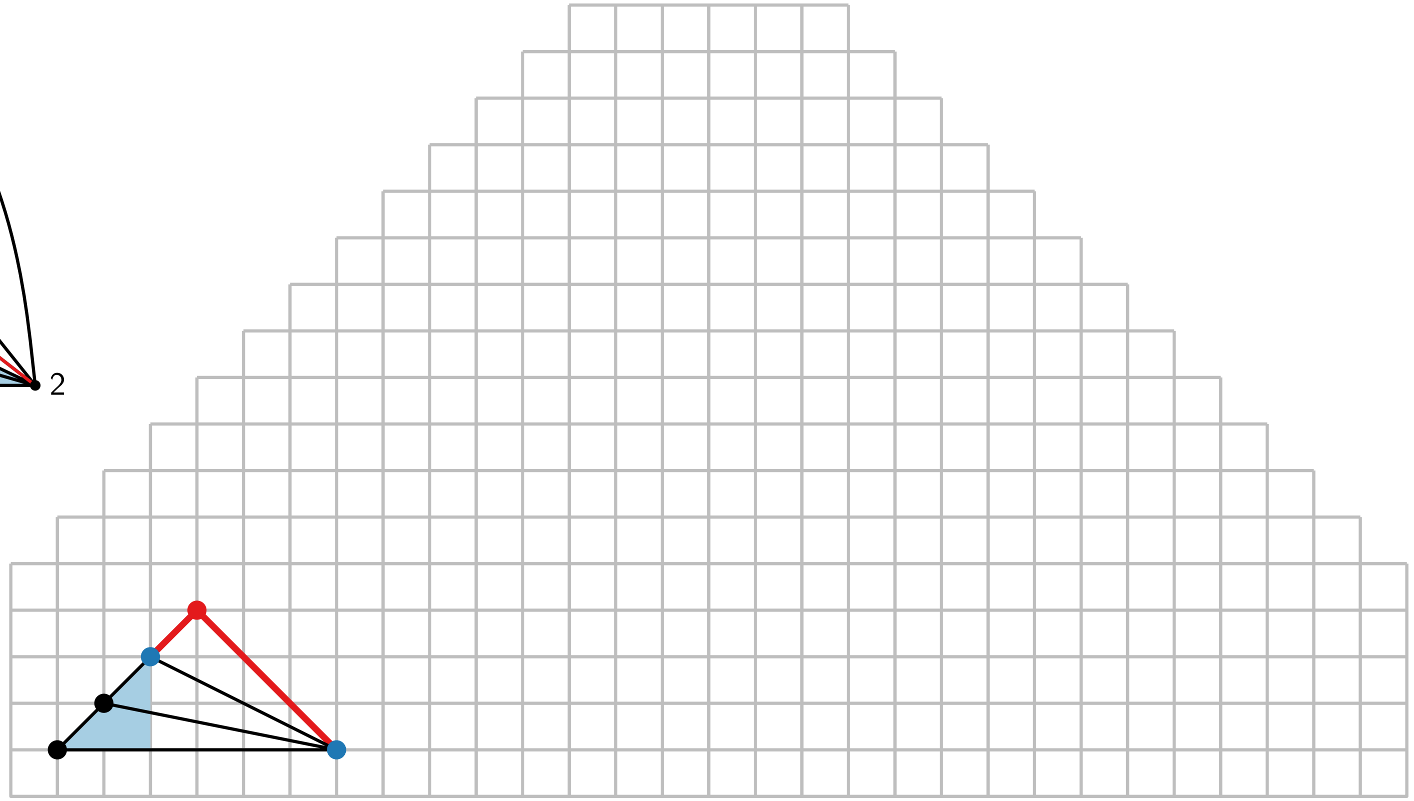




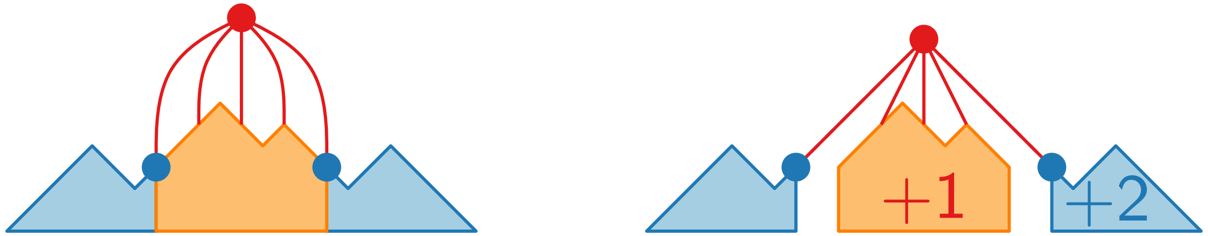
Shift method – example

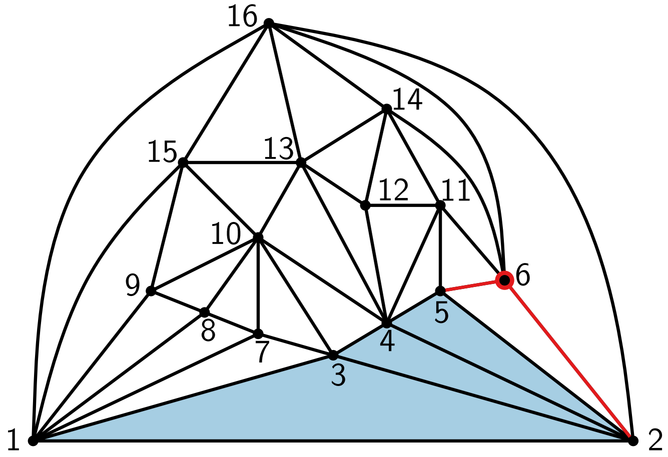


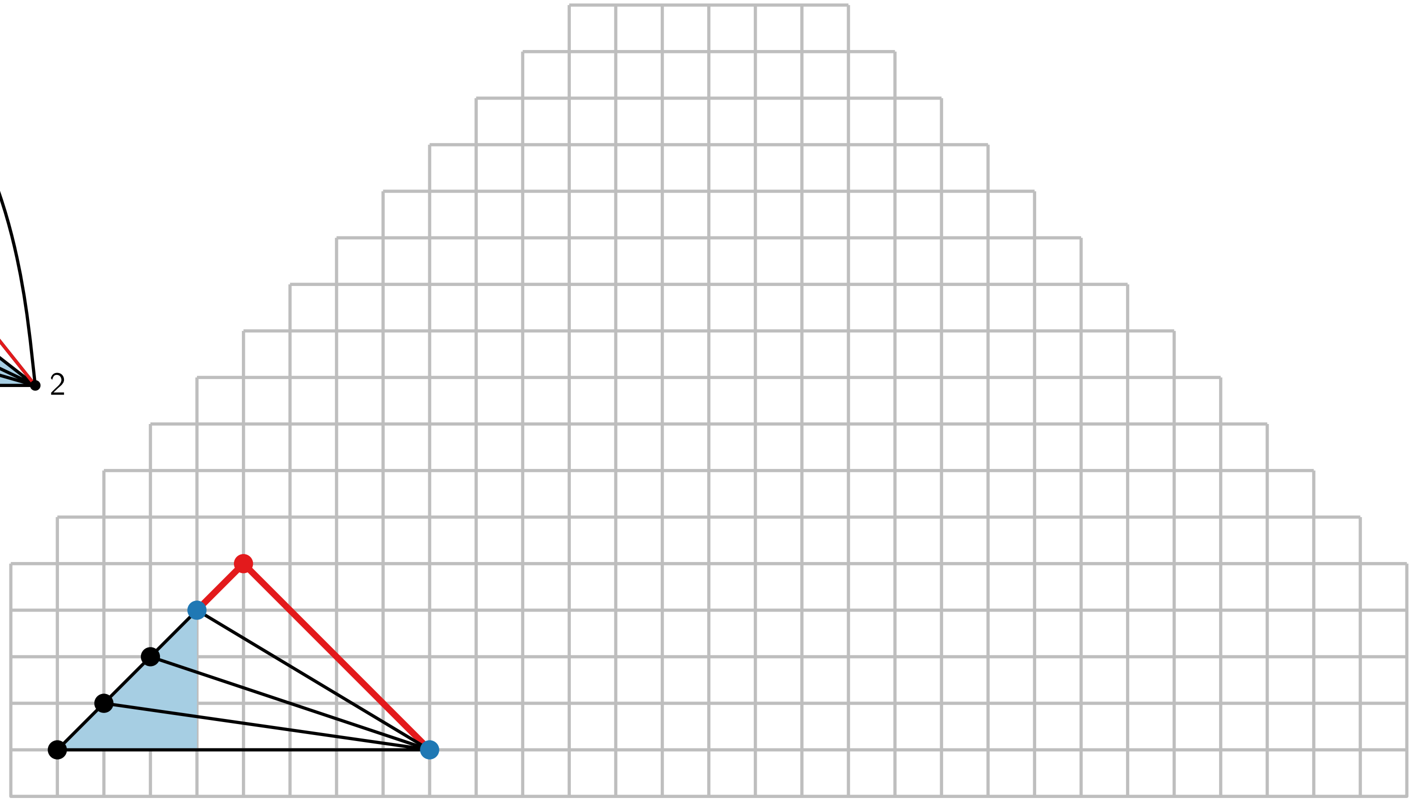




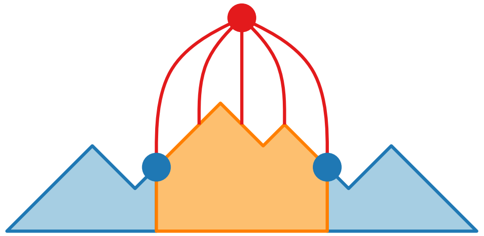
Shift method – example

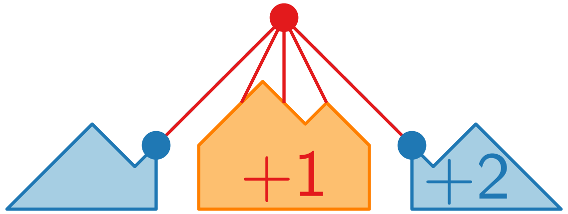


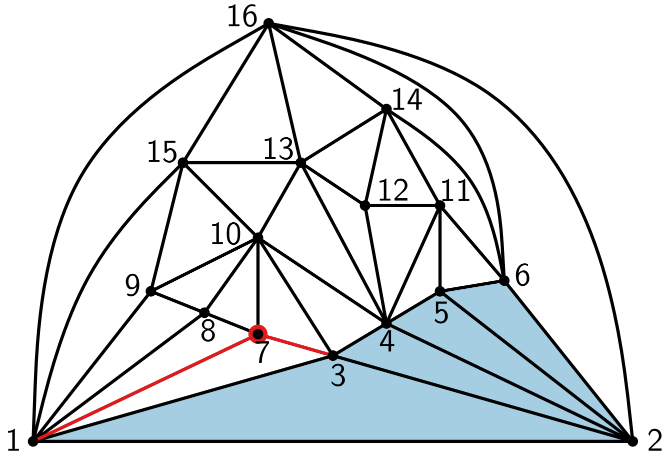


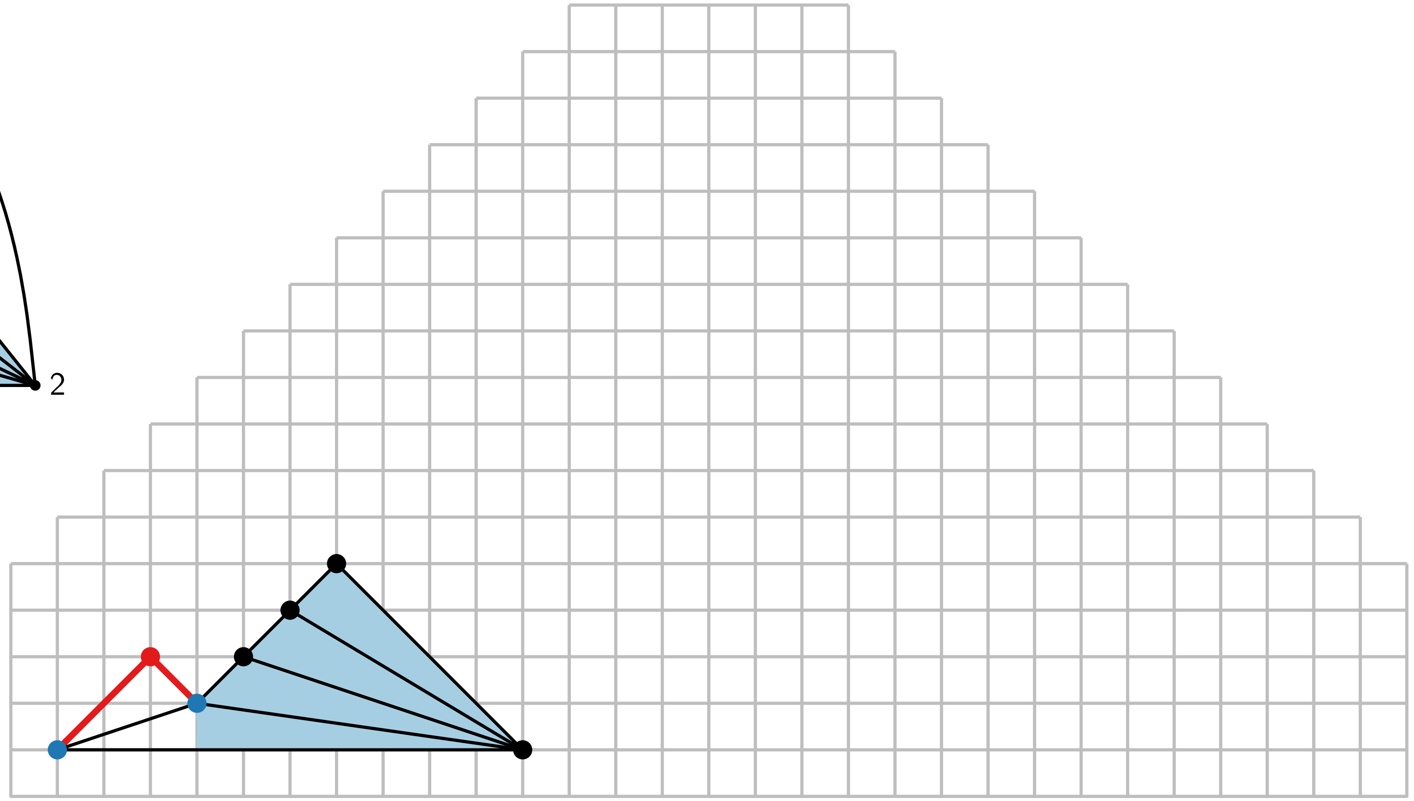


Shift method – example

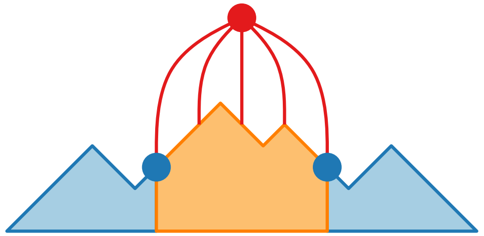


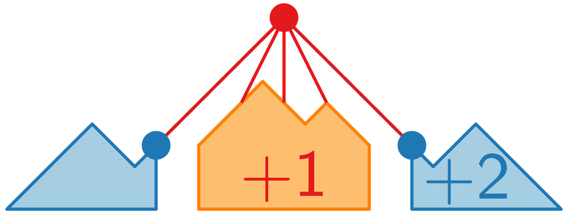


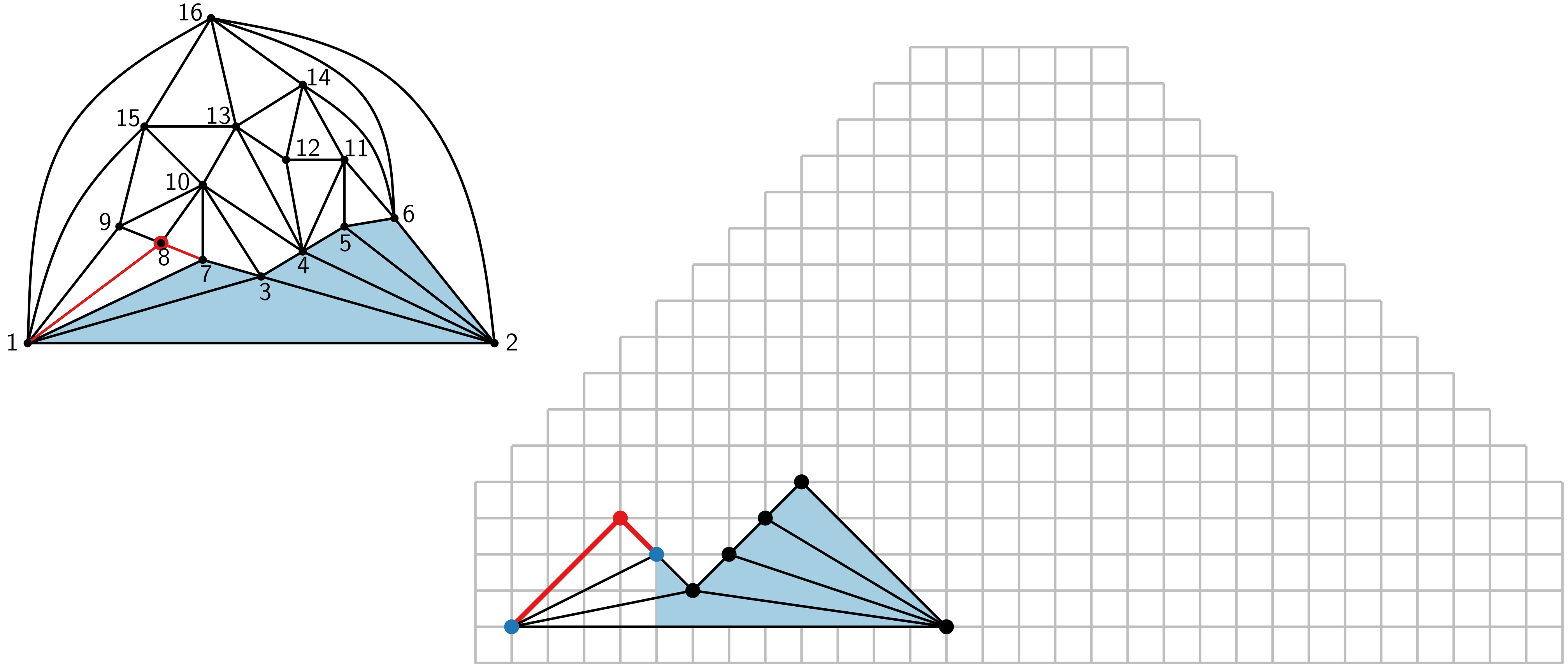




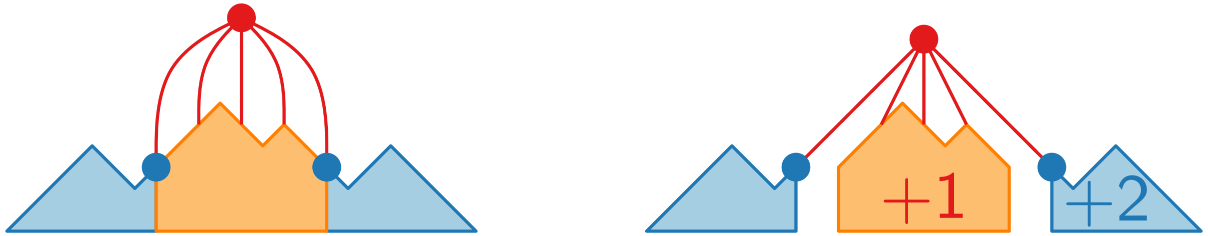
Shift method – example

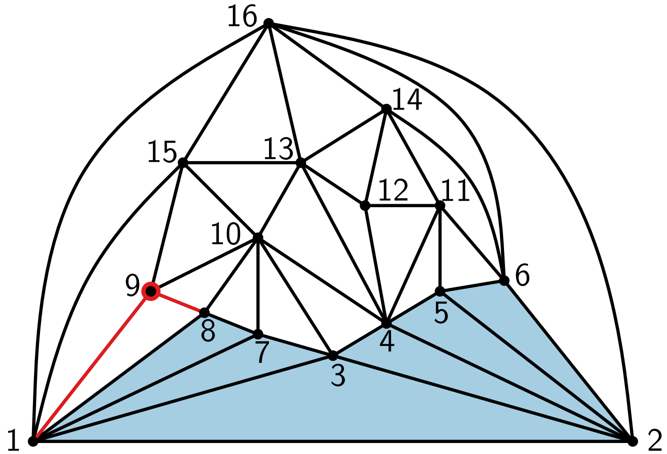


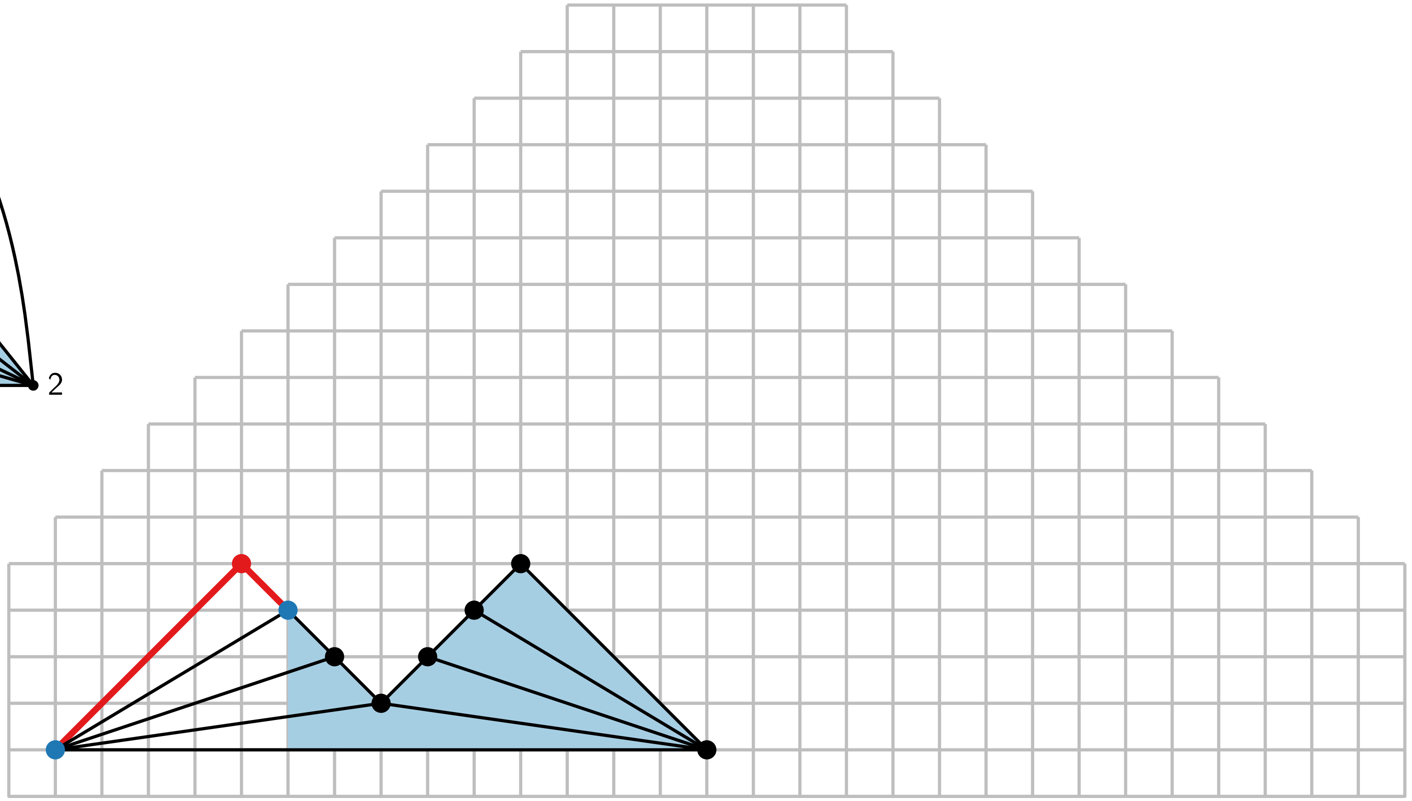




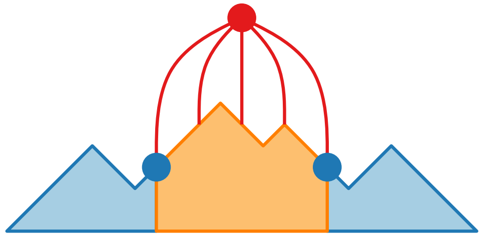
Shift method – example

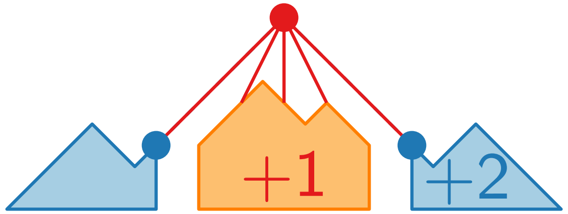


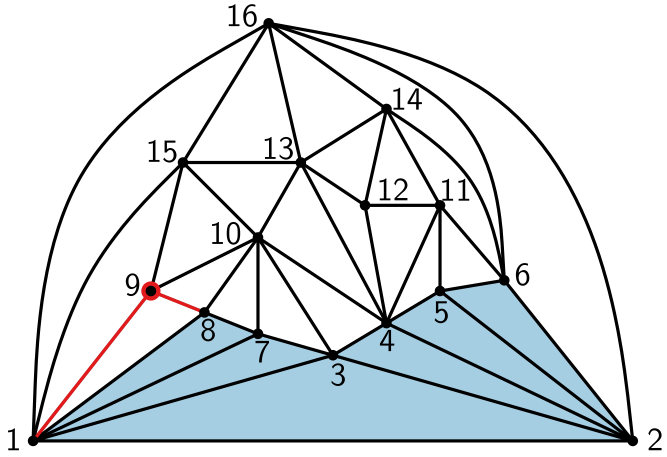


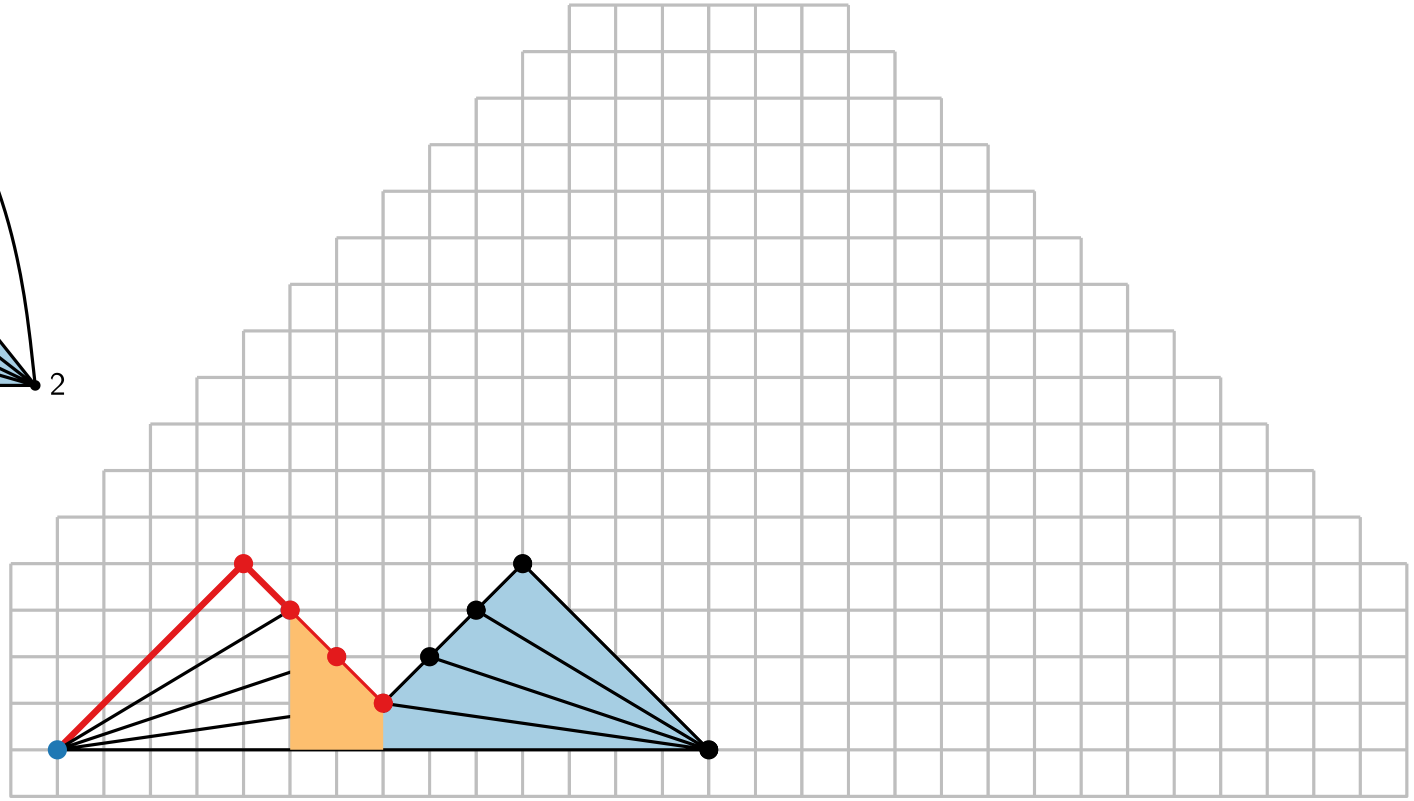


Shift method – example

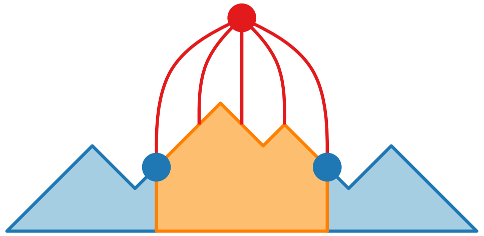


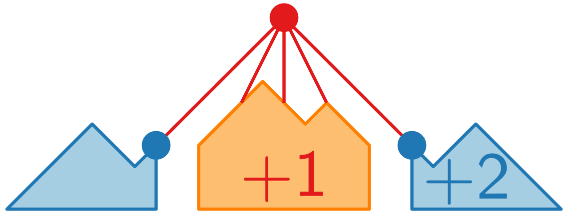


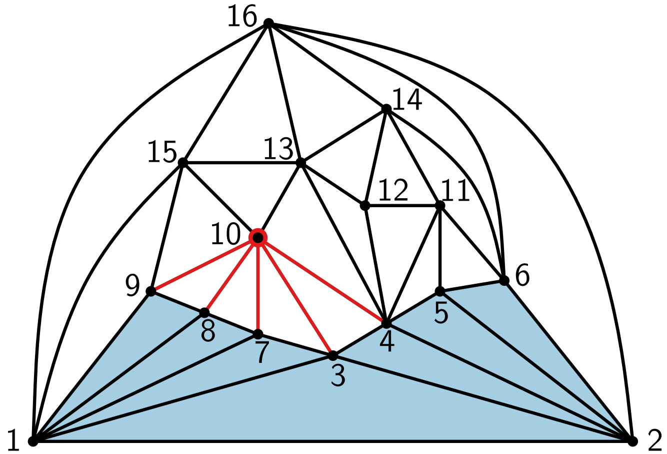


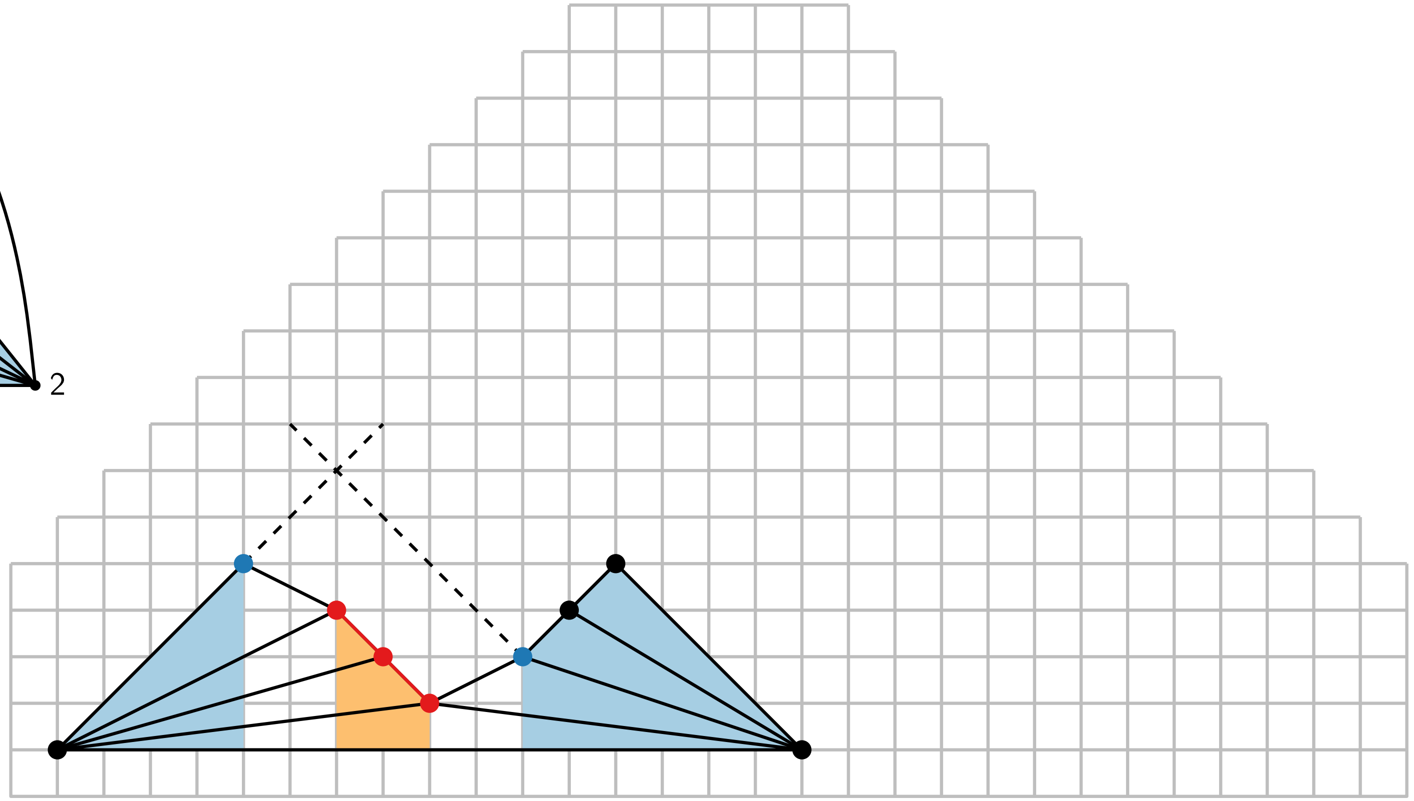


Shift method – example

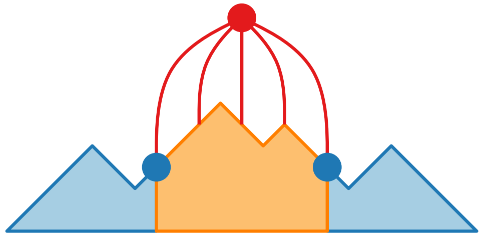


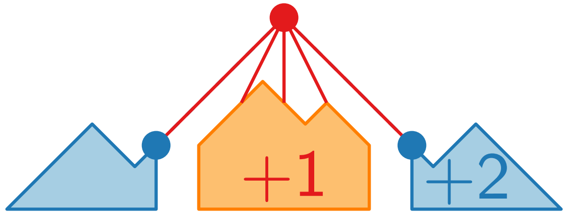


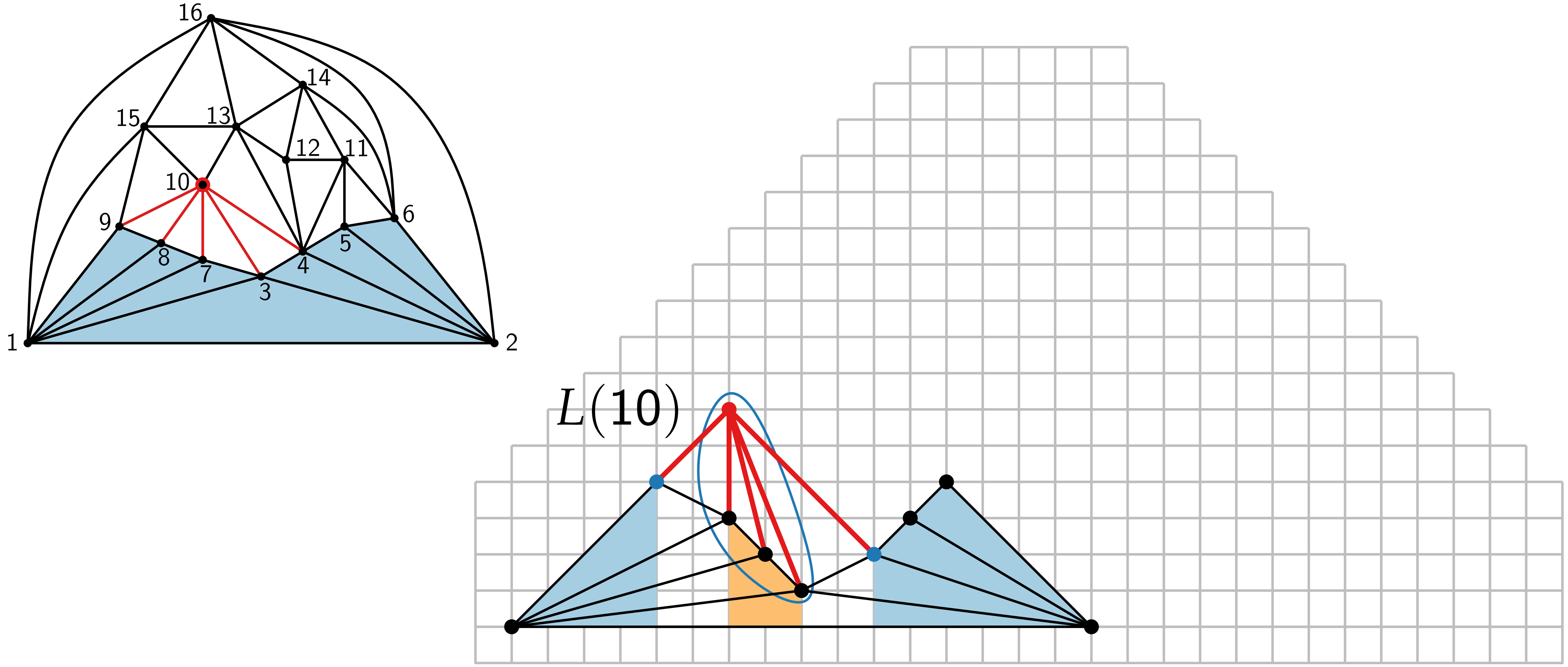




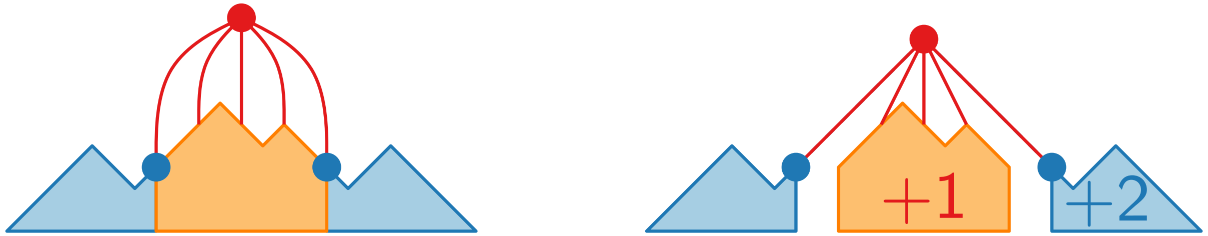
Shift method – example

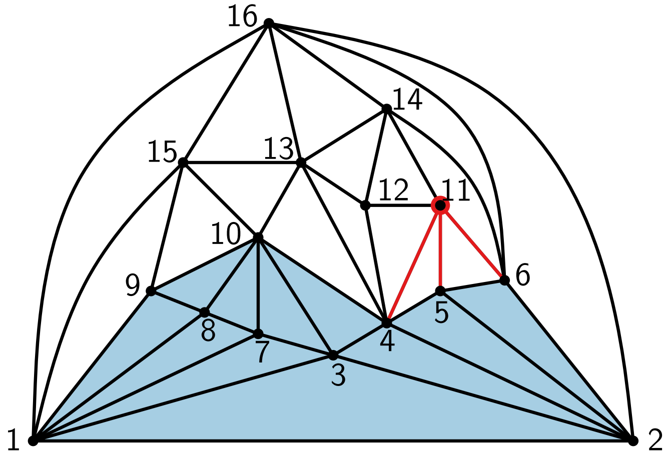


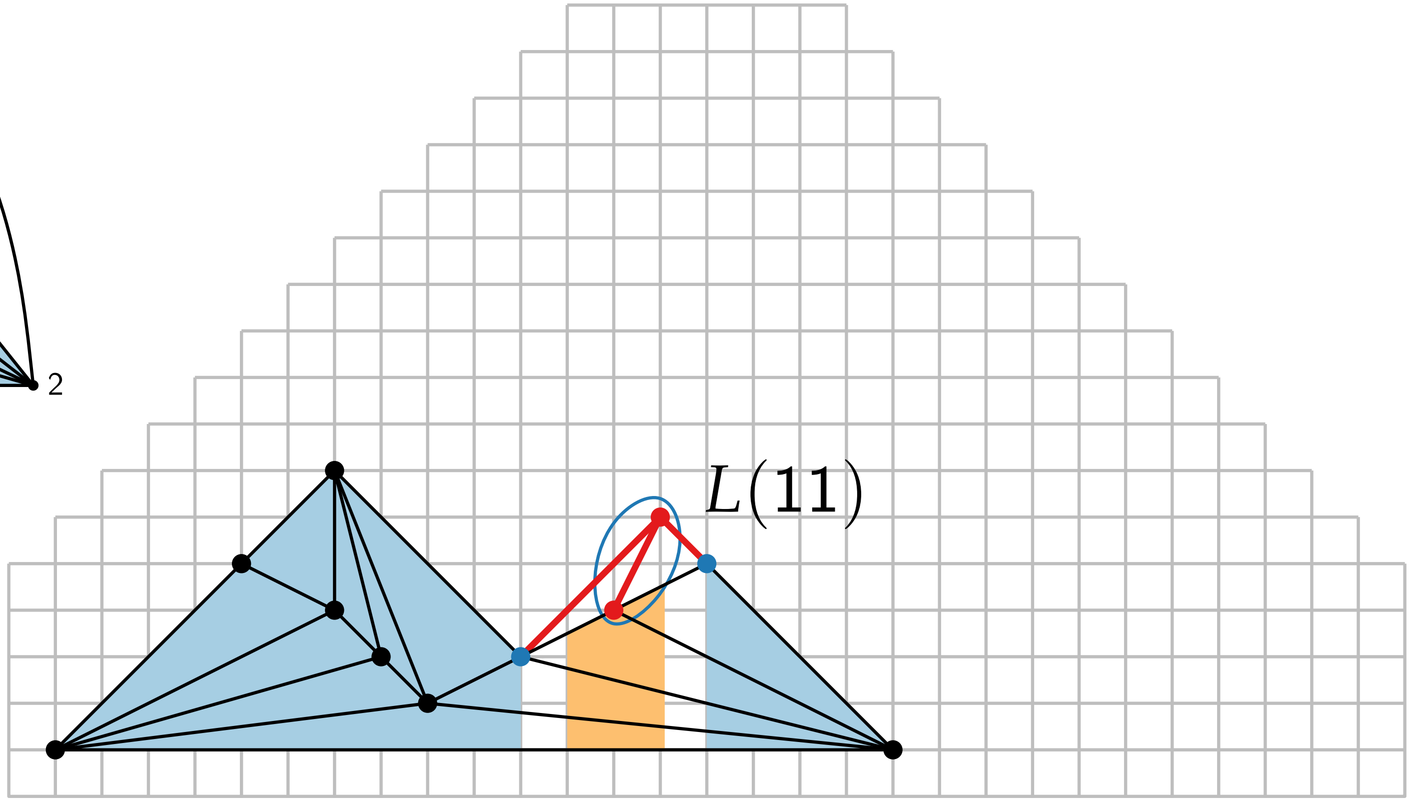




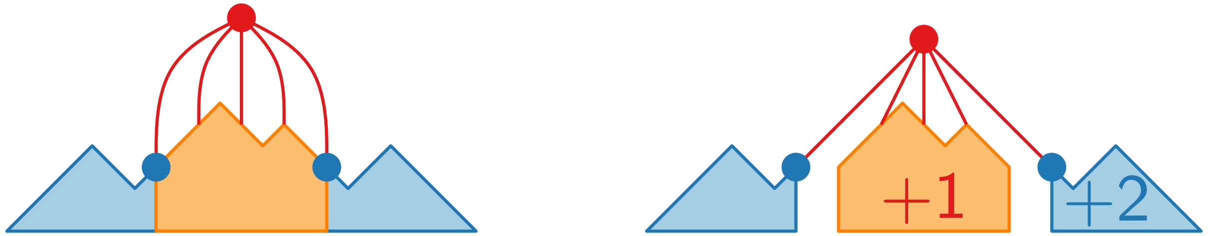
Shift method – example

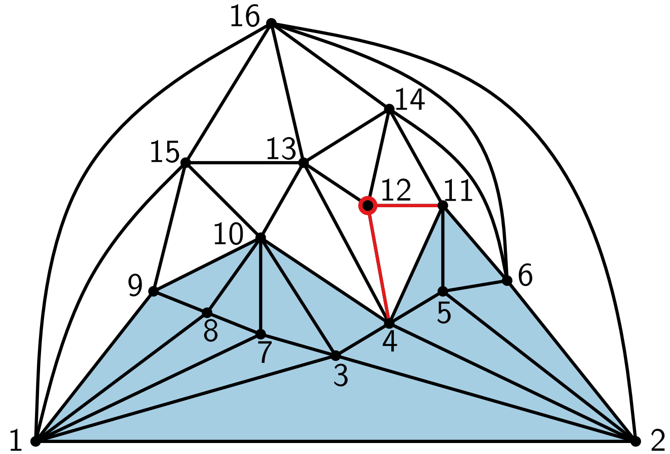


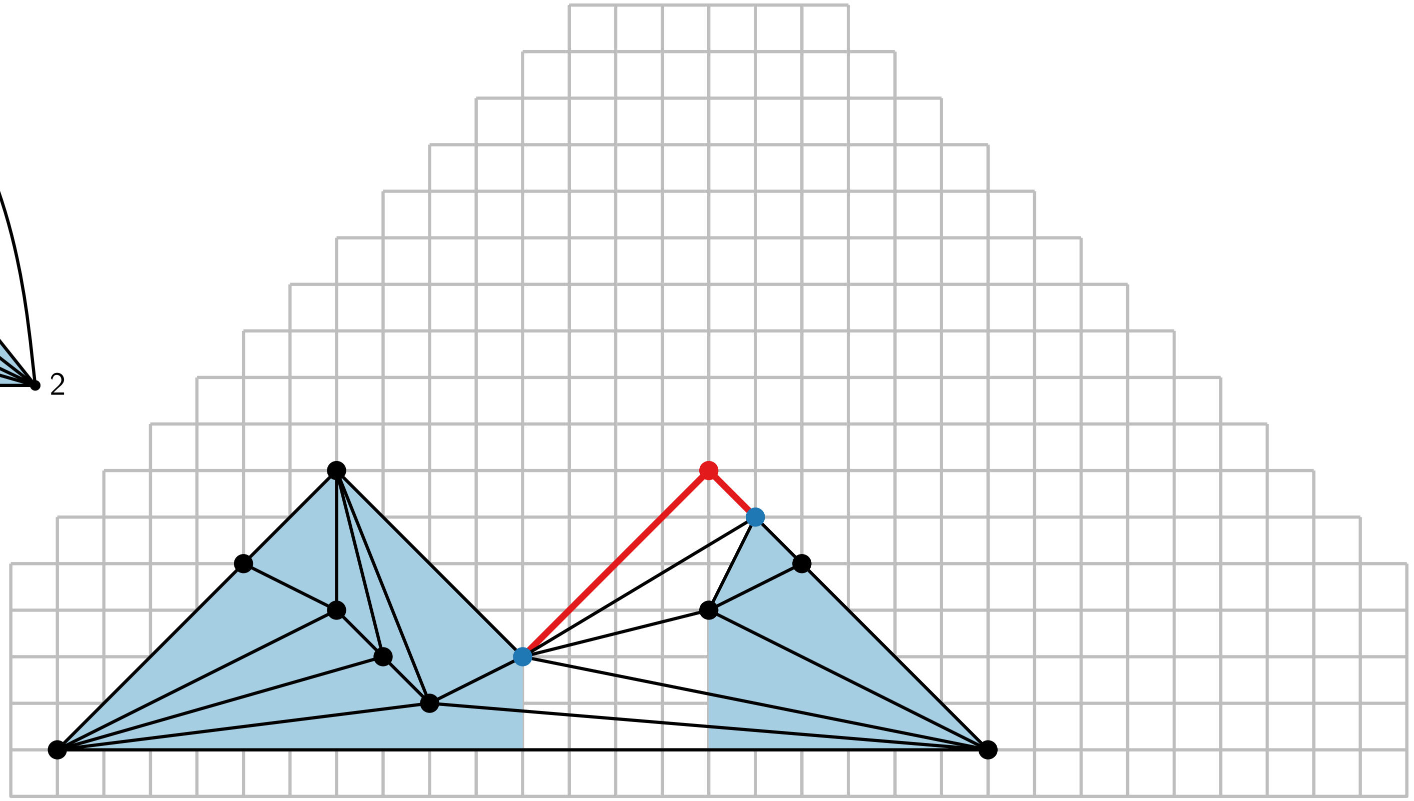




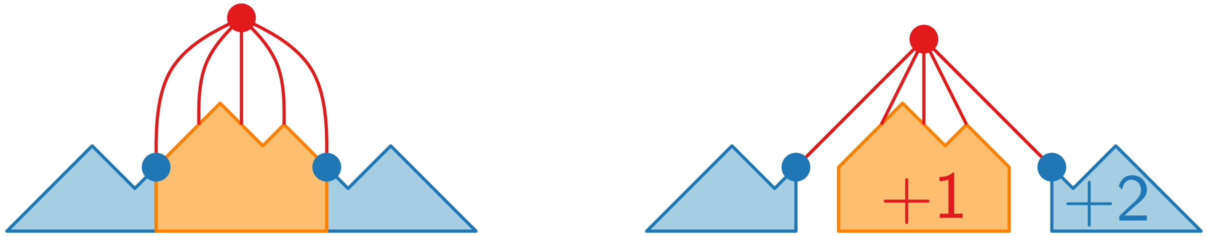
Shift method – example

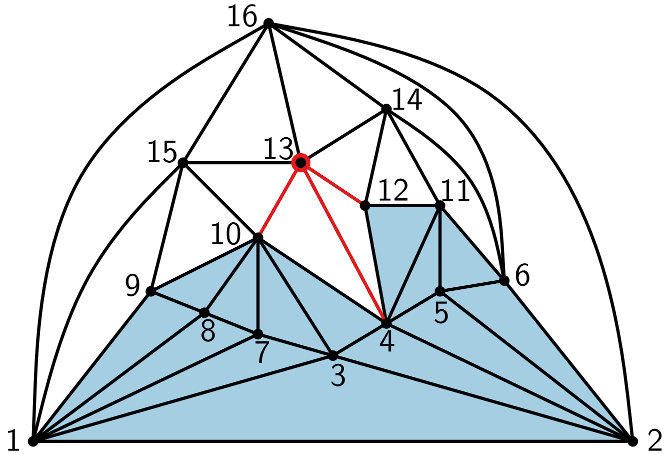


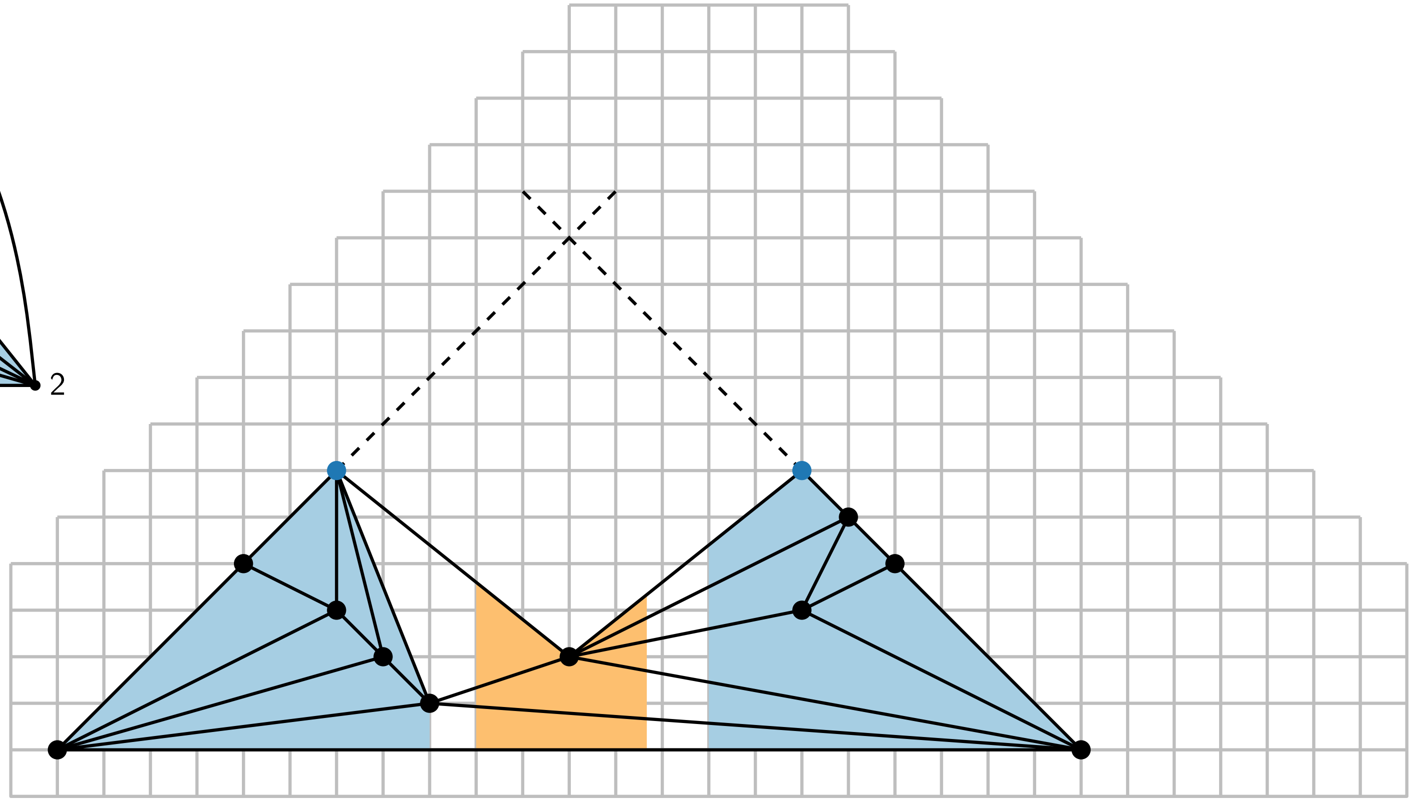




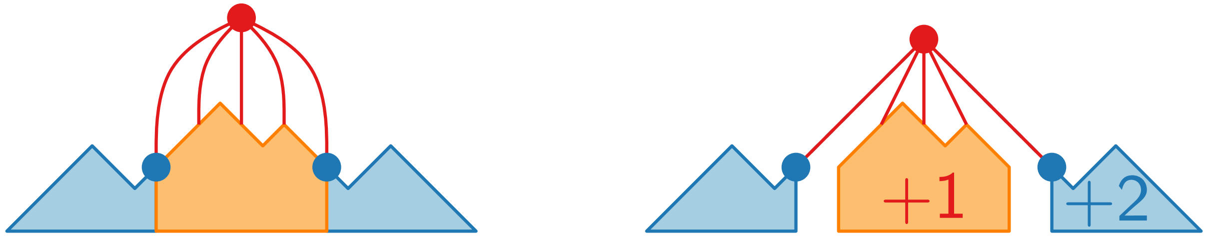
Shift method – example

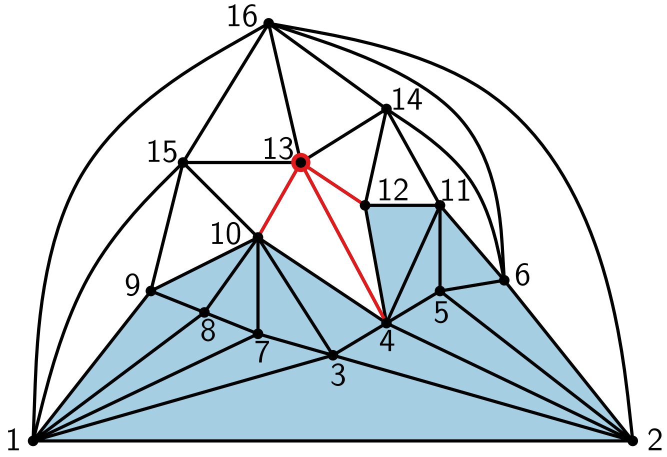


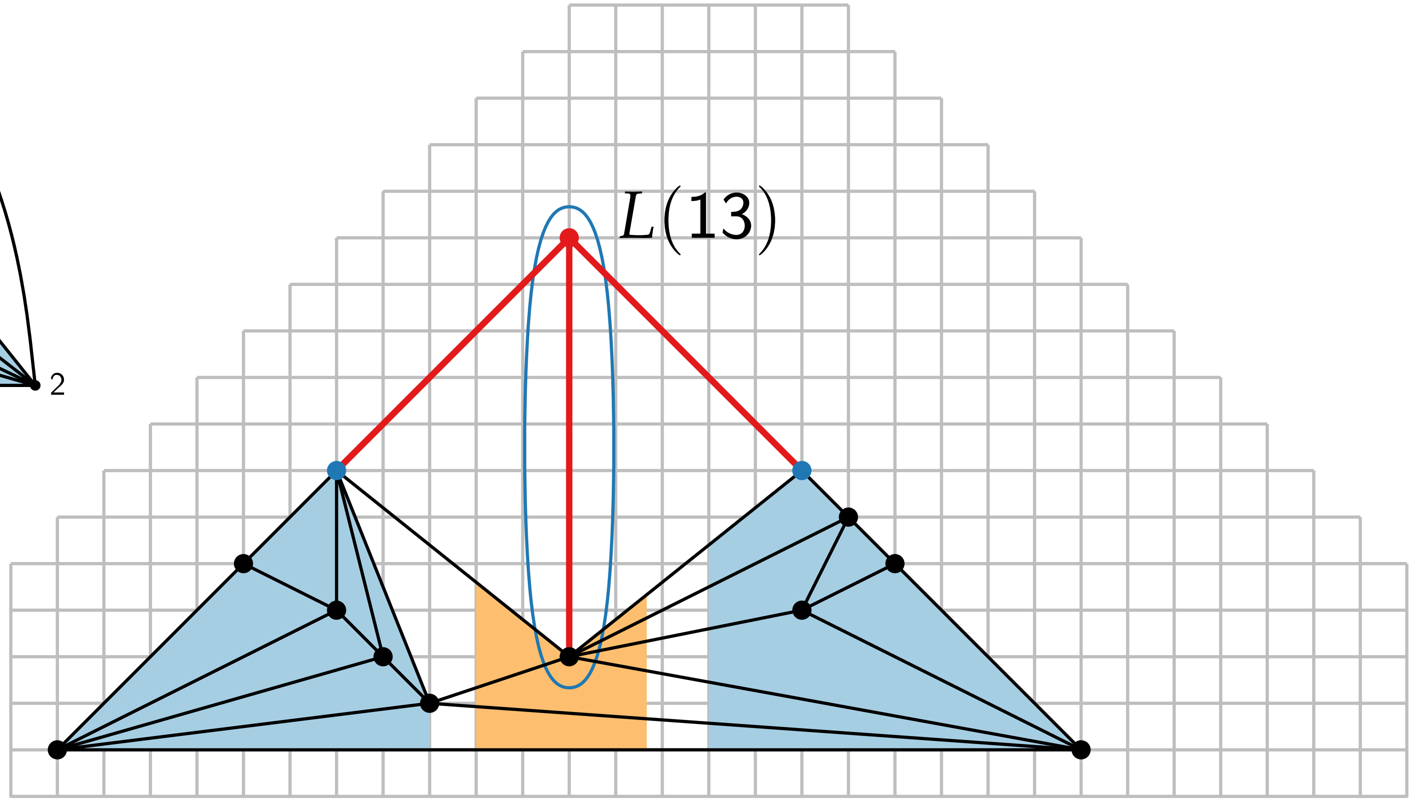




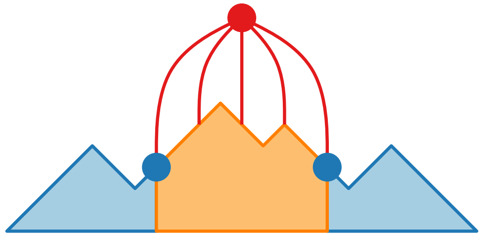
Shift method – example

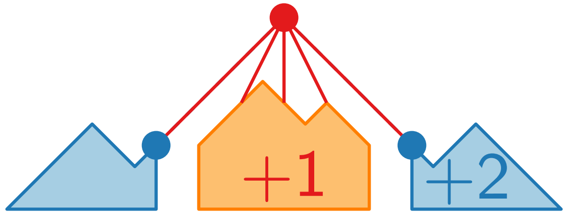


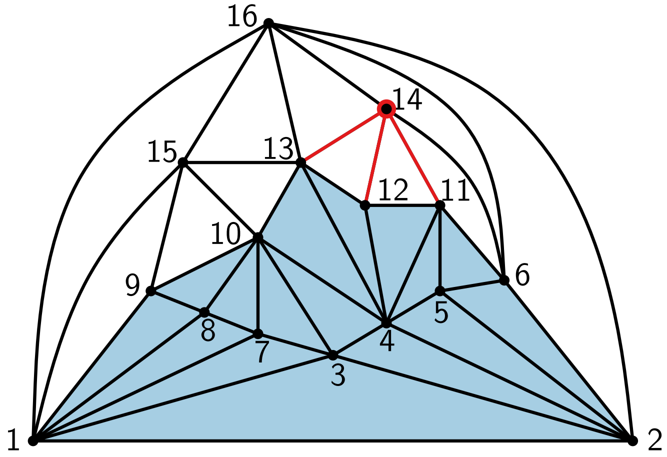


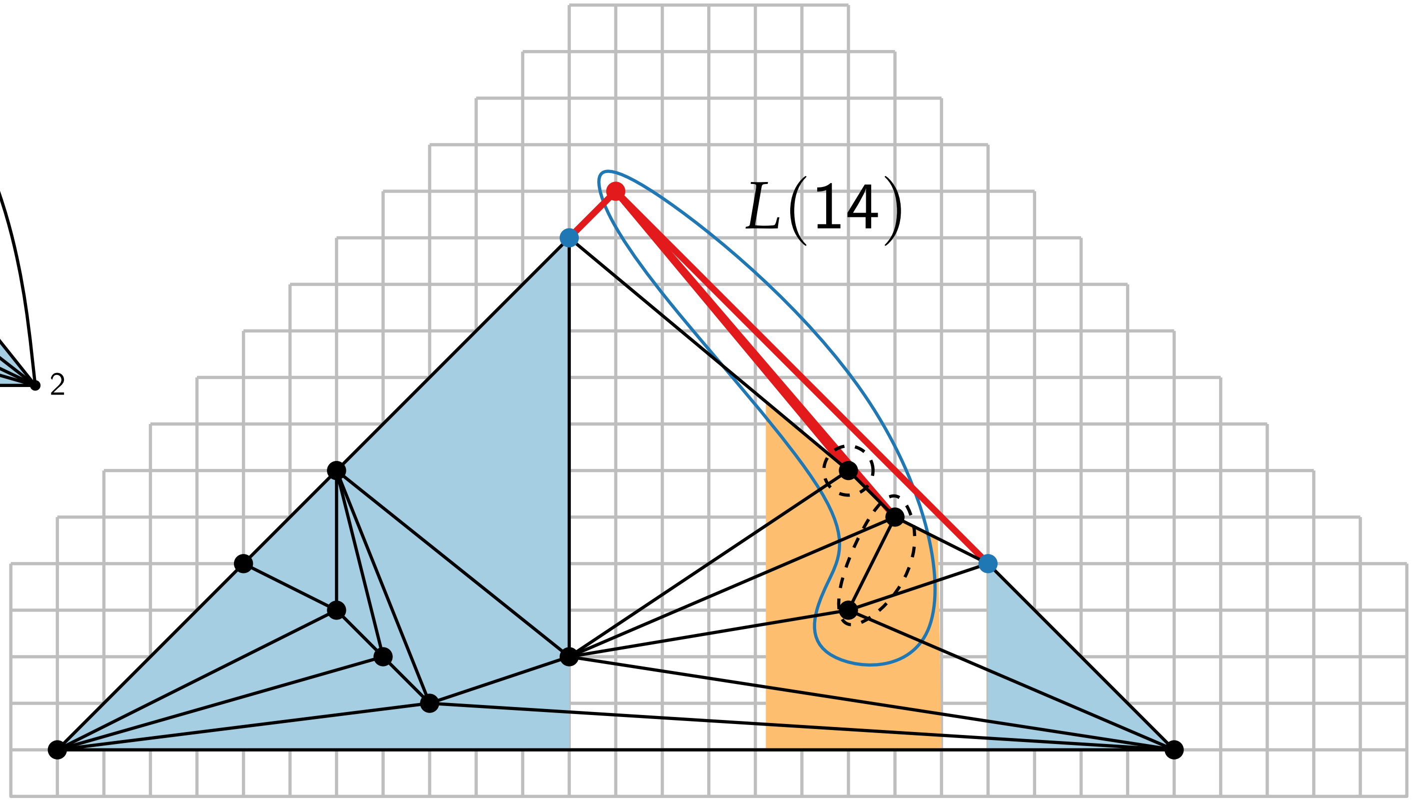


Shift method – example

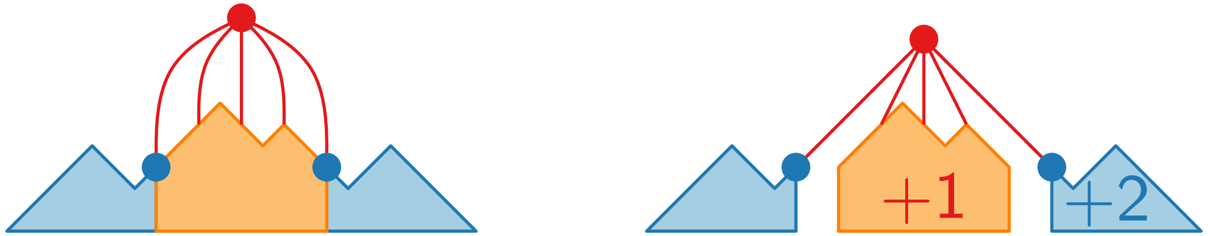


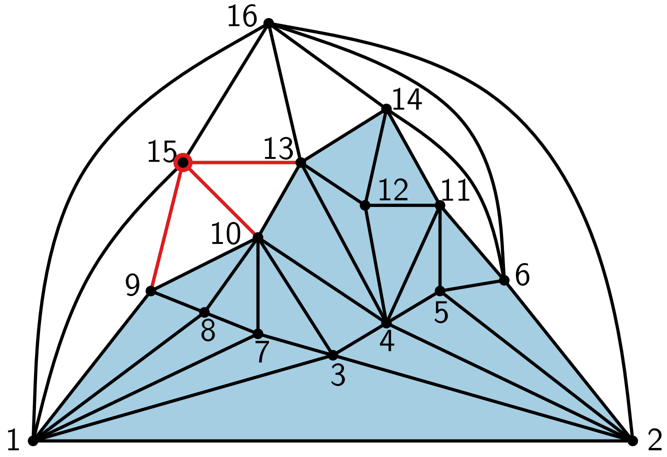


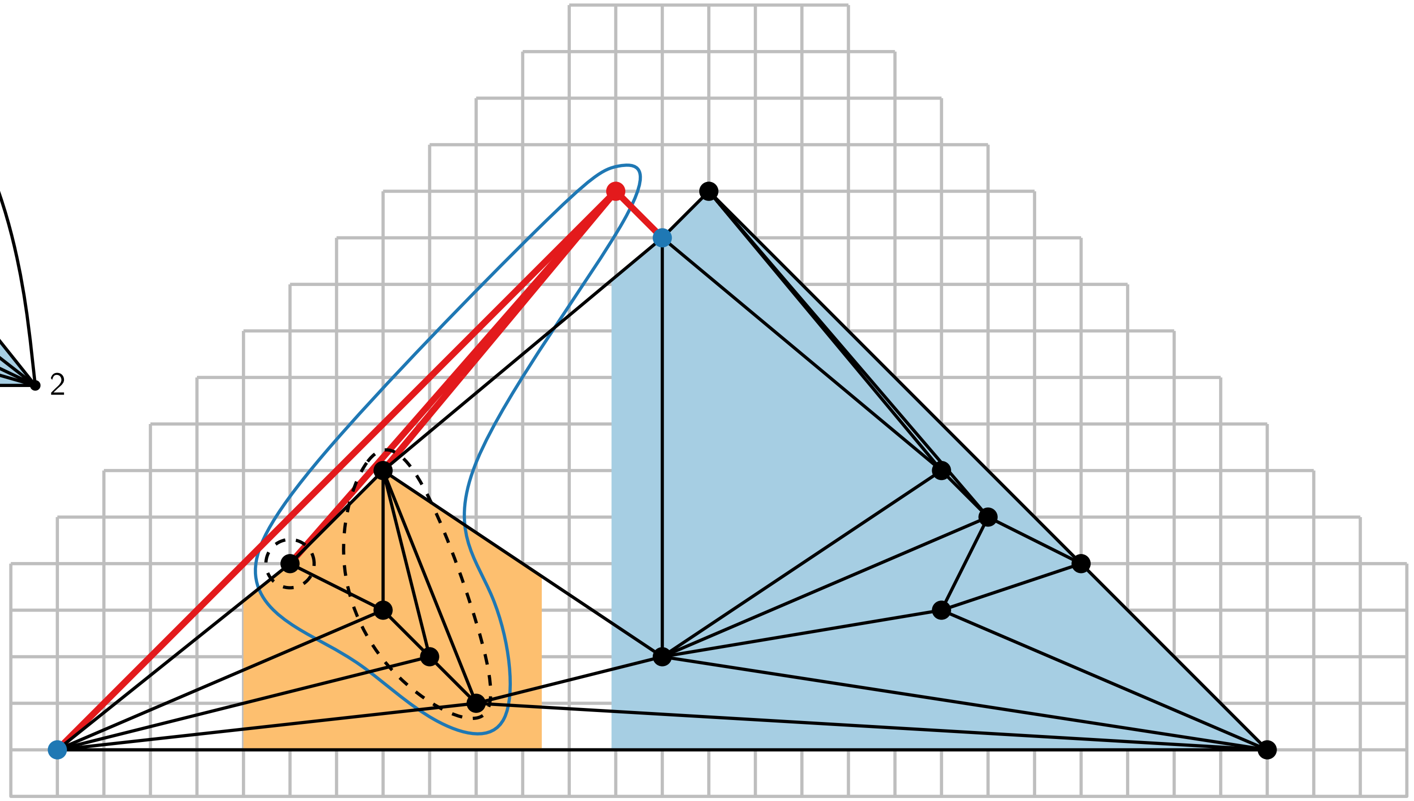




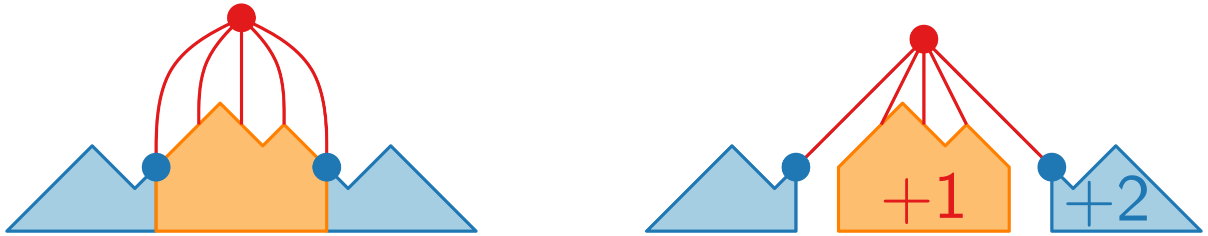
Shift method – example

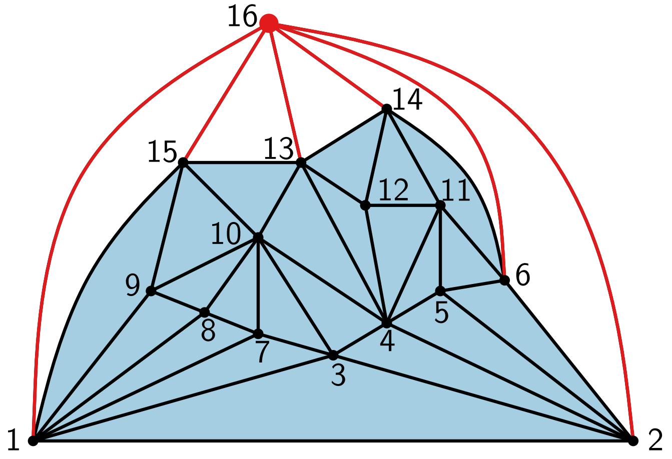


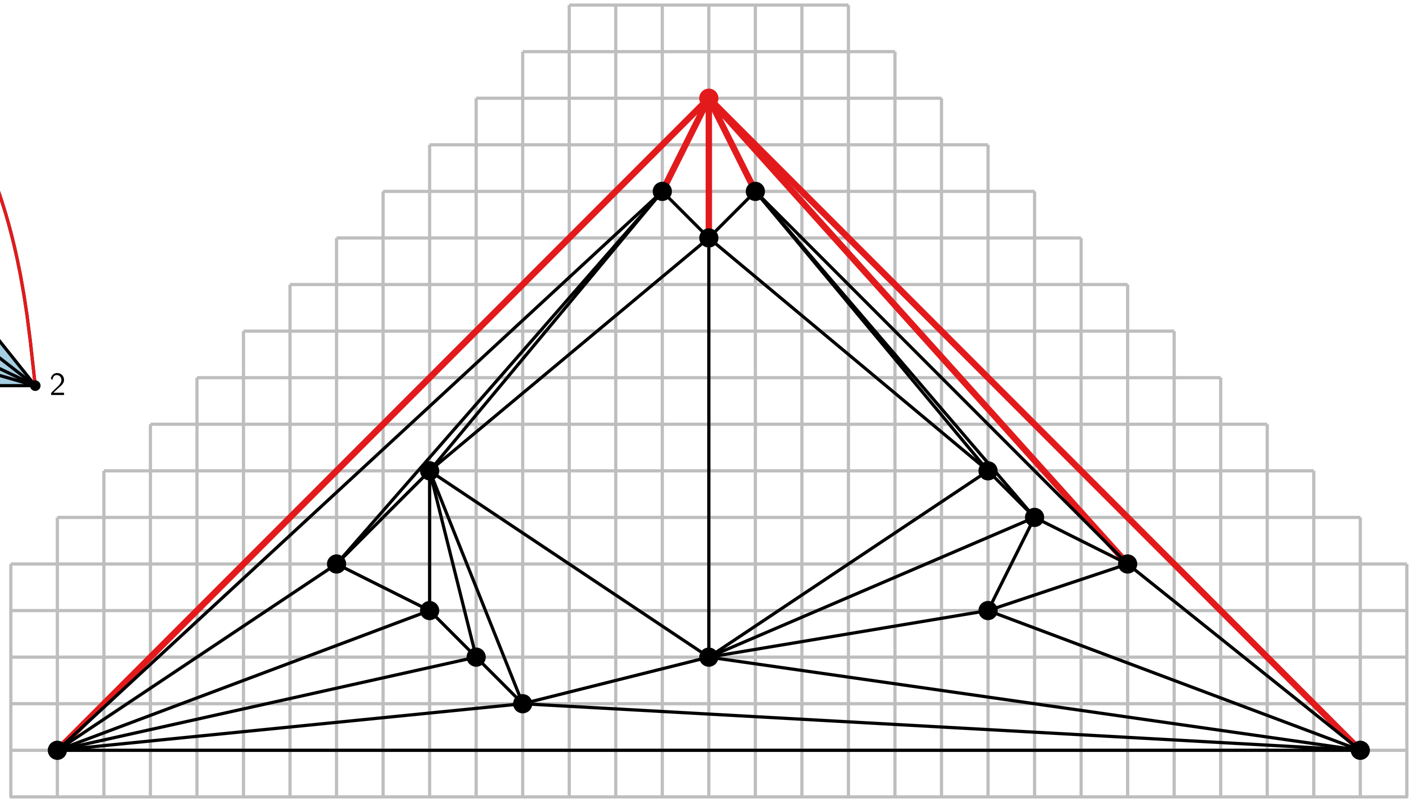




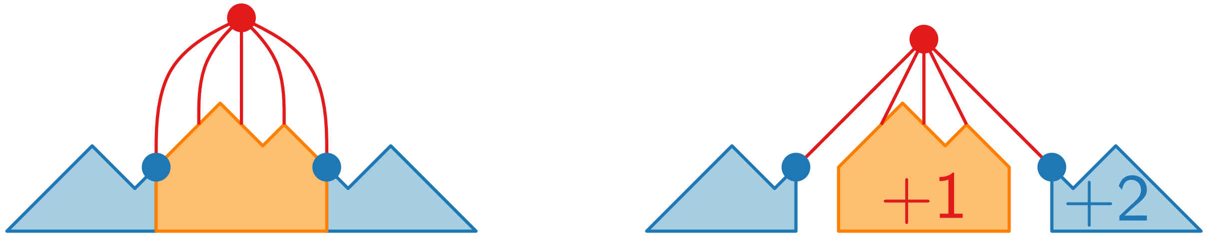
Shift method – example

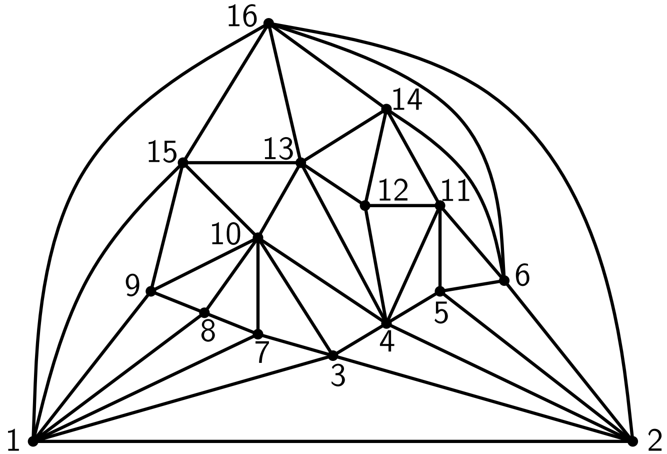


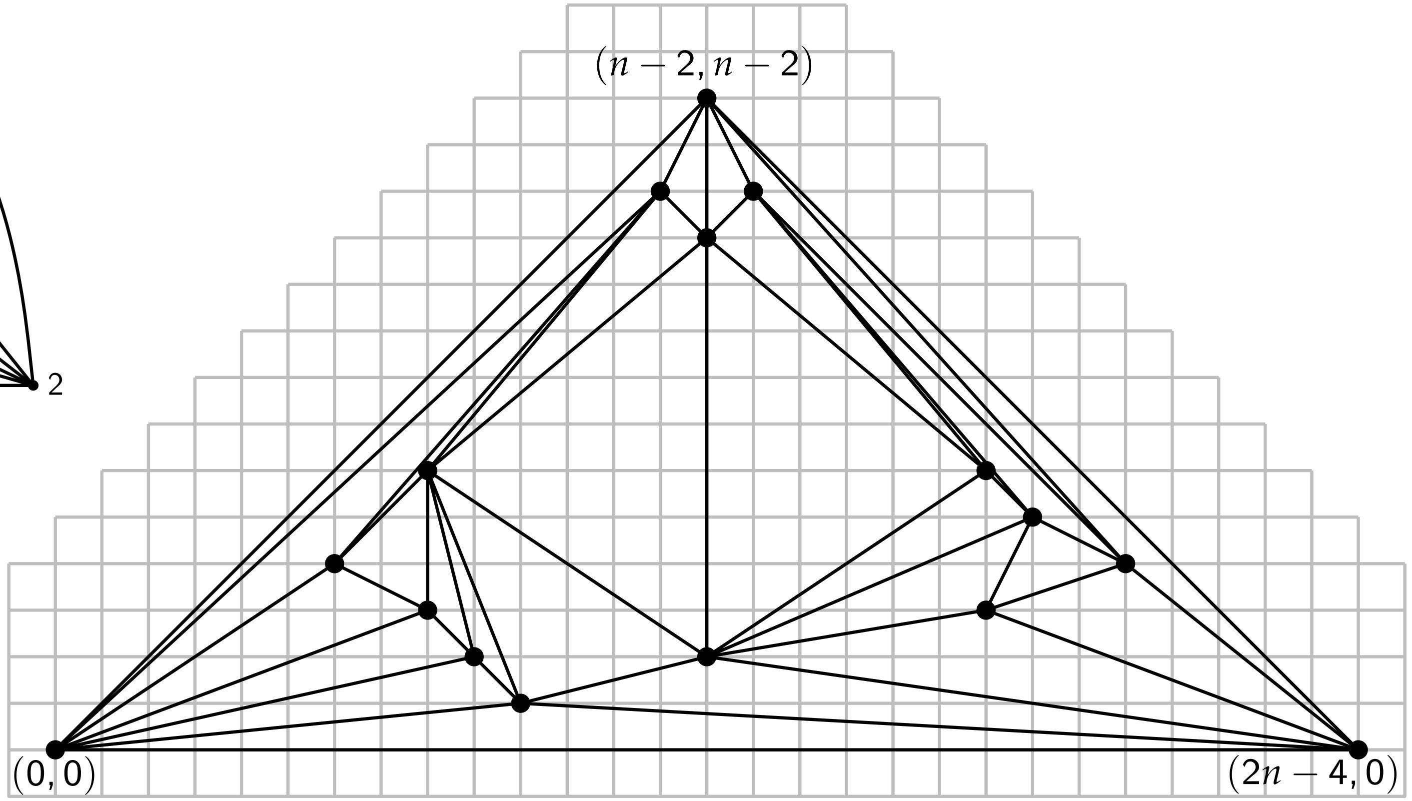




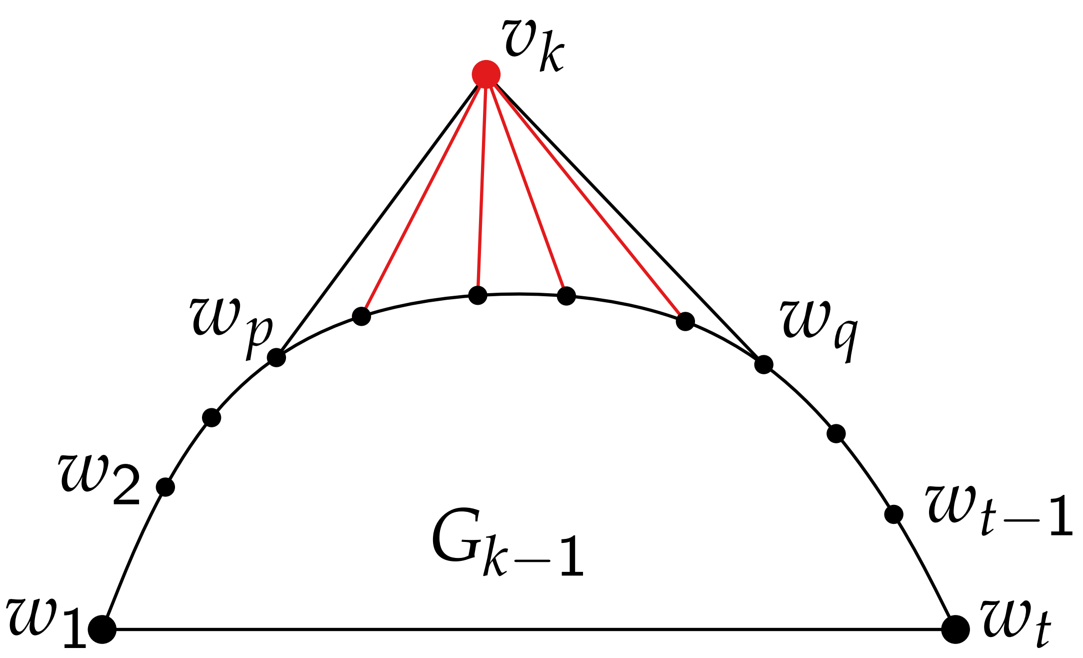
Shift method – example



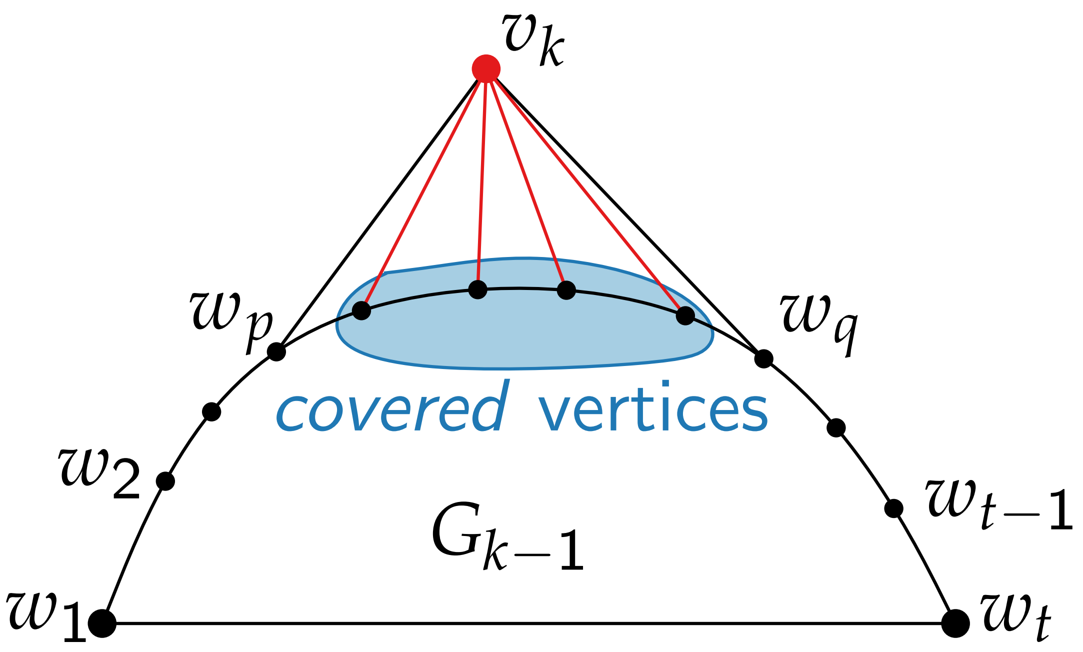




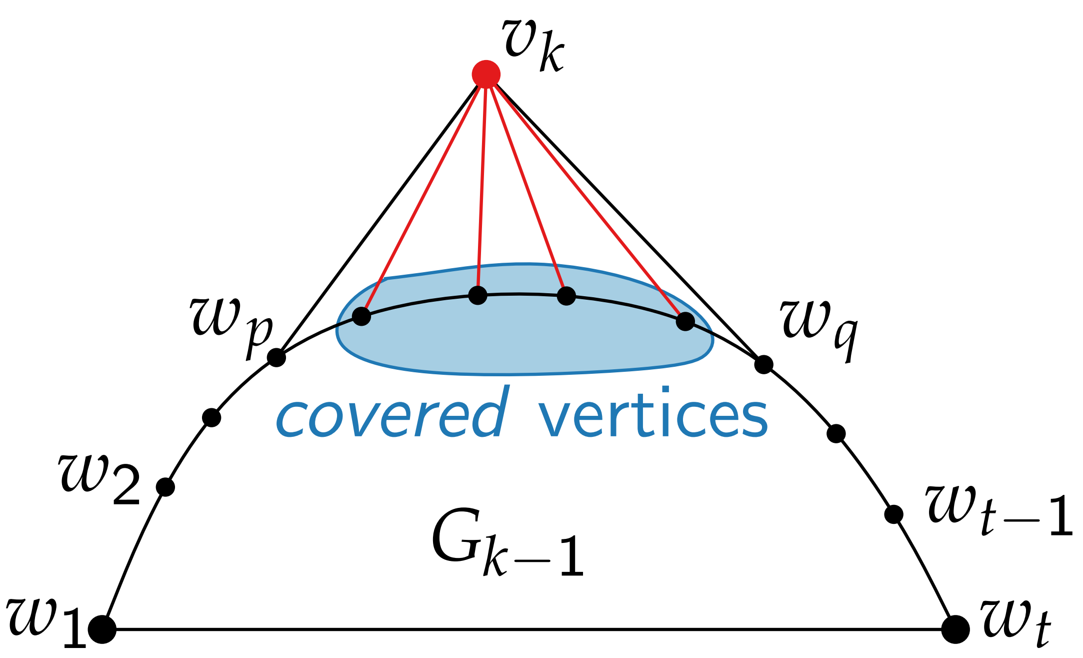
Shift method – planarity



Shift method – planarity



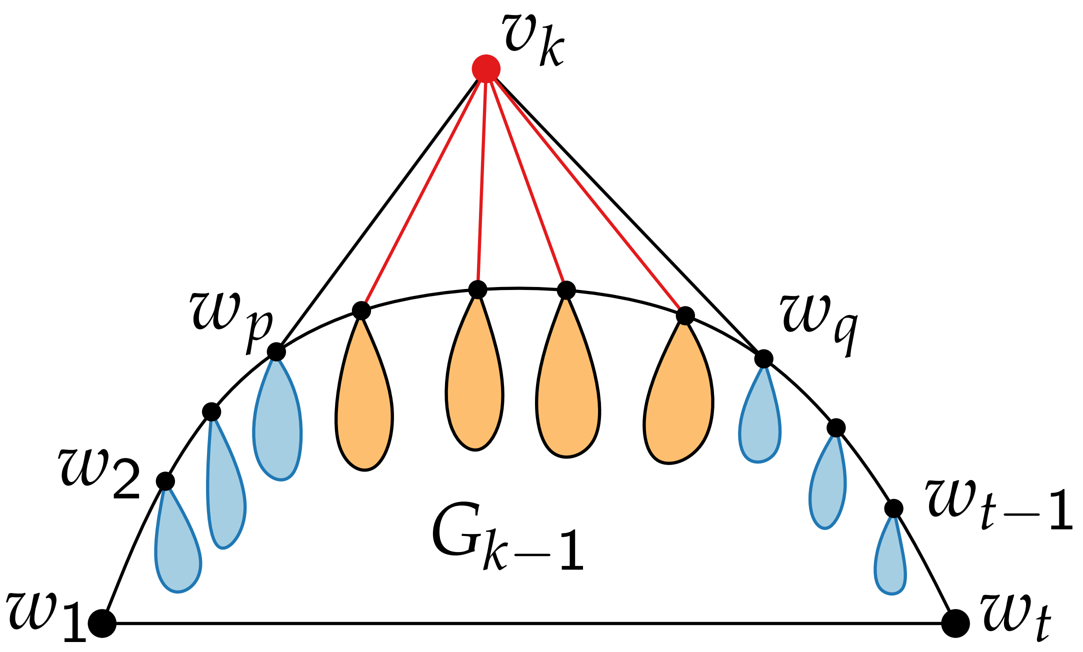
Shift method – planarity



Observations.

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

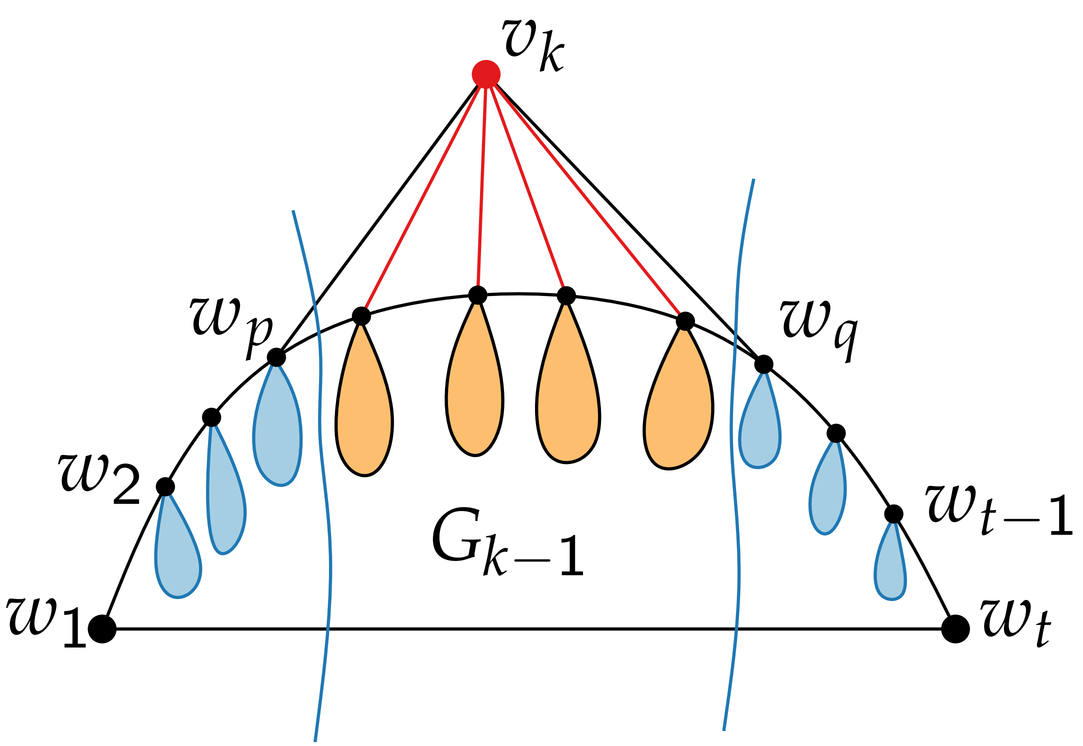
Shift method – planarity



Observations.

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

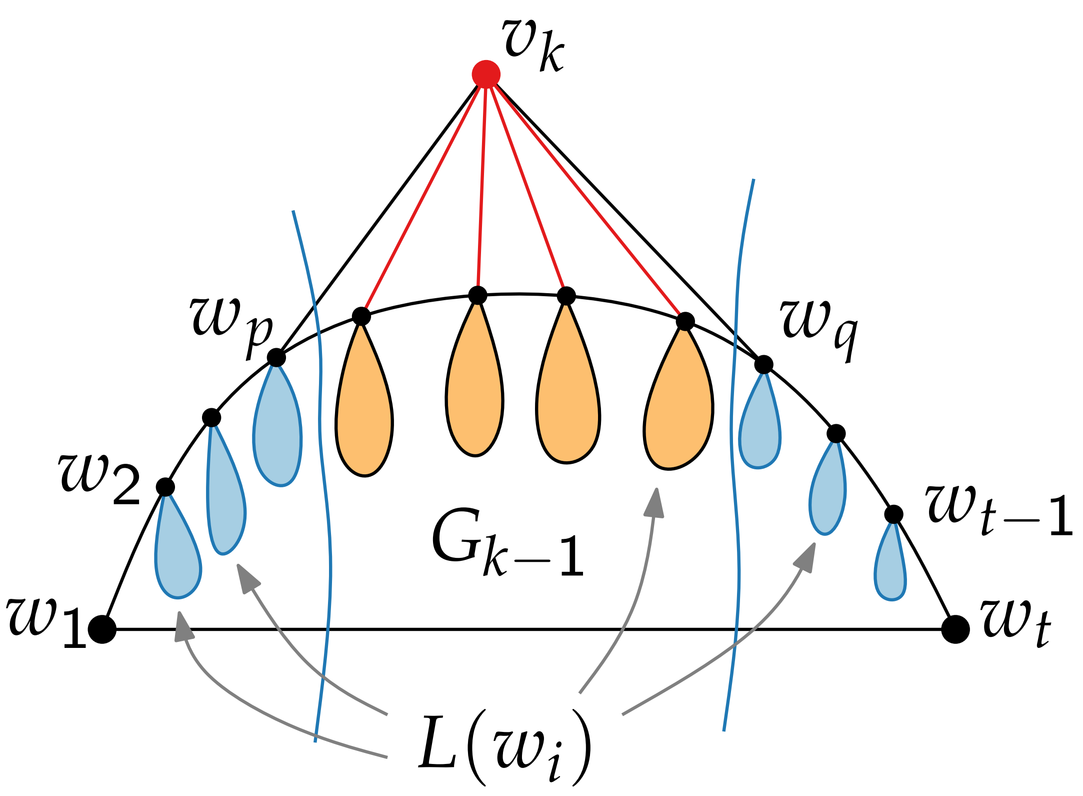
Shift method – planarity



Observations.

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

Shift method – planarity

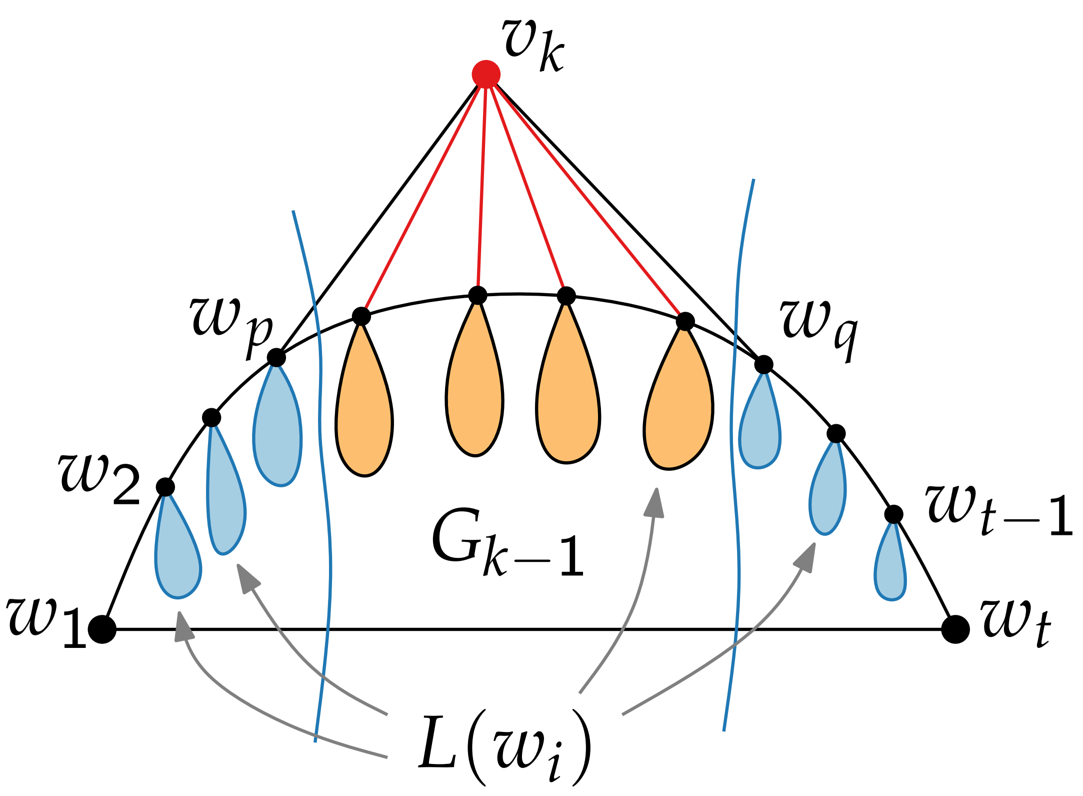


Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \le i \le n-1$.

Shift method – planarity

Lemma. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, such that $\delta_q - \delta_p \geq 2$ and even.
If we shift $L(w_i)$ by δ_i to the right, we get a planar straight-line drawing.



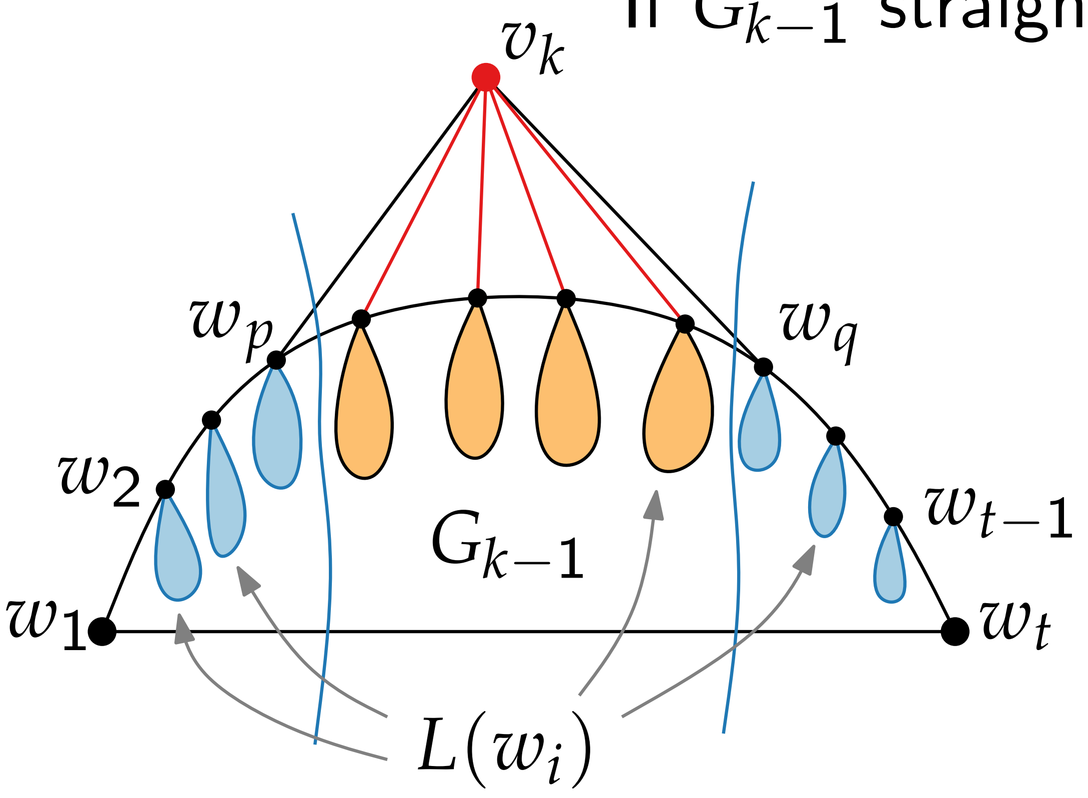
Observations.

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

Shift method – planarity

Lemma. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, such that $\delta_q - \delta_p \geq 2$ and even.
If we shift $L(w_i)$ by δ_i to the right, we get a planar straight-line drawing.

Proof by induction:
If G_{k-1} straight-line planar, then also G_k.



Observations.

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

Shift method – pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

for $i = 1$ to 3 **do**

  $L(v_i) \leftarrow \{v_i\}$

$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$

for $i = 4$ to n **do**

  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{i-1} and let $w_p, \ldots, w_q$ be the neighbours of v_k

  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**

    $x(v) \leftarrow x(v) + 1$

  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do**

    $x(v) \leftarrow x(v) + 2$

  $P(v_i) \leftarrow$ intersection of $+1/-1$ edges from $P(w_p)$ and $P(w_q)$

  $L(v_i) \leftarrow \cup_{j=p+1}^{q-i} L(w_j) \cup \{v_i\}$

Shift method – pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

for $i = 1$ to 3 **do**

  $L(v_i) \leftarrow \{v_i\}$

$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$

for $i = 4$ to n **do**

  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{i-1} and let $w_p, \ldots, w_q$ be the neighbours of v_k

  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**

    $x(v) \leftarrow x(v) + 1$

  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do**

    $x(v) \leftarrow x(v) + 2$

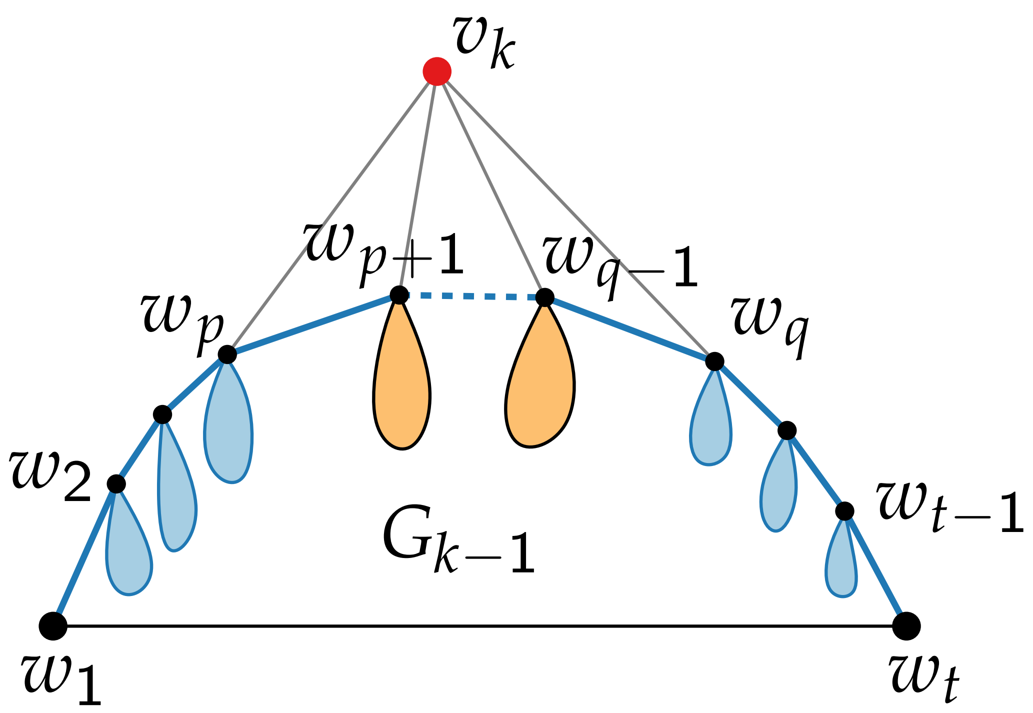
  $P(v_i) \leftarrow$ intersection of $+1/-1$ edges from $P(w_p)$ and $P(w_q)$

  $L(v_i) \leftarrow \cup_{j=p+1}^{q-i} L(w_j) \cup \{v_i\}$

- Runtime $\mathcal{O}(n^2)$
- Can we do better?

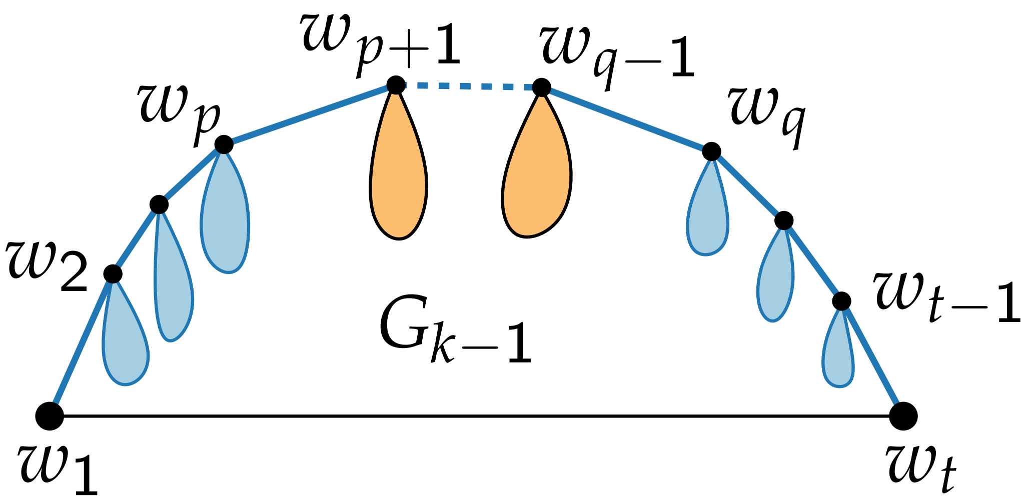
Shift method – linear time implementation

- **Idea 1.** To compute $x(v_k)$ & $y(v_k)$, we only need $y(w_p)$ and $y(w_q)$ and $x(w_q) - x(w_p)$



Shift method – linear time implementation

- **Idea 1.** To compute $x(v_k)$ & $y(v_k)$, we only need $y(w_p)$ and $y(w_q)$ and $x(w_q) - x(w_p)$



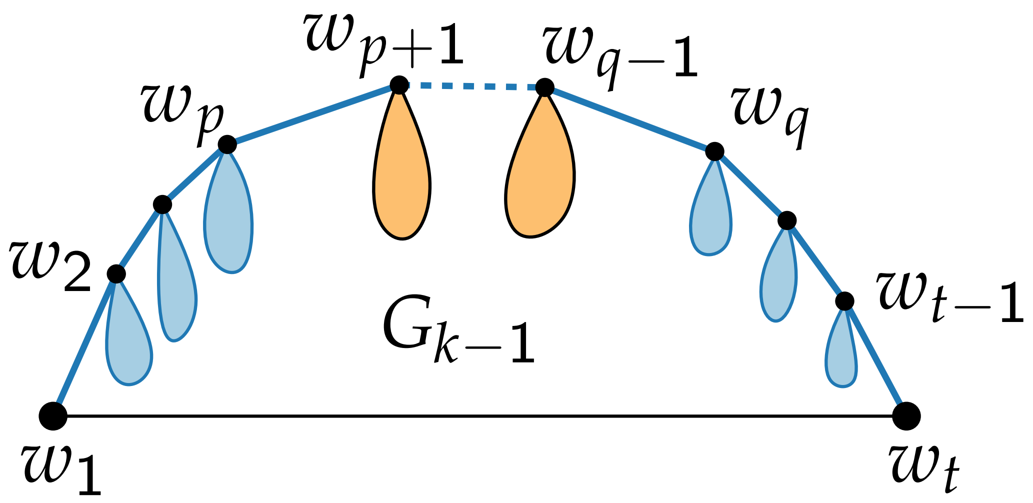
(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

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

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

Shift method – linear time implementation

- **Idea 1.** To compute $x(v_k)$ & $y(v_k)$, we only need $y(w_p)$ and $y(w_q)$ and $x(w_q) - x(w_p)$
- **Idea 2.** Instead of storing explicit x-coordinates, we store certain x differences.



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

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

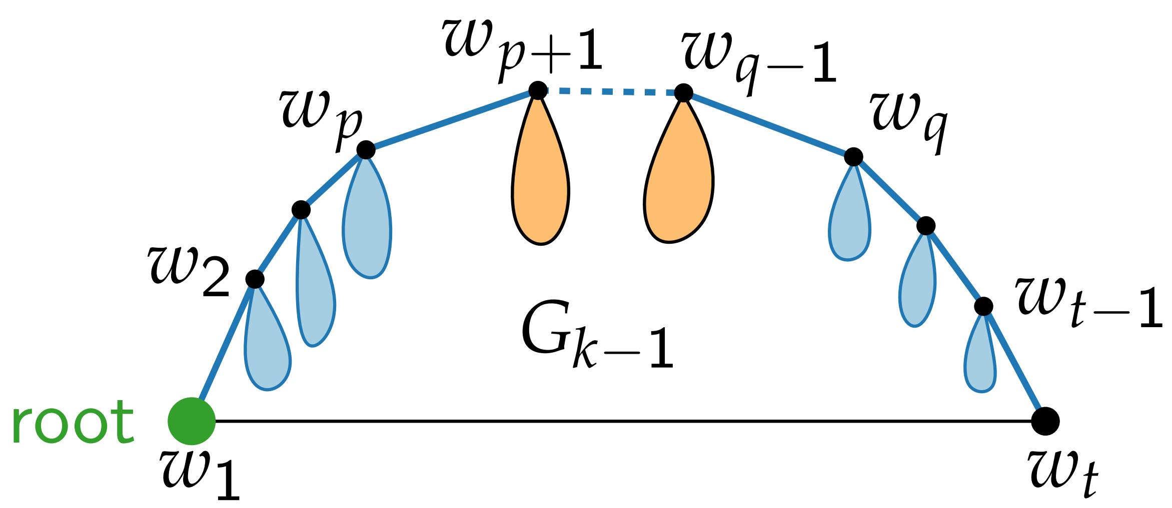
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

Relative x distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

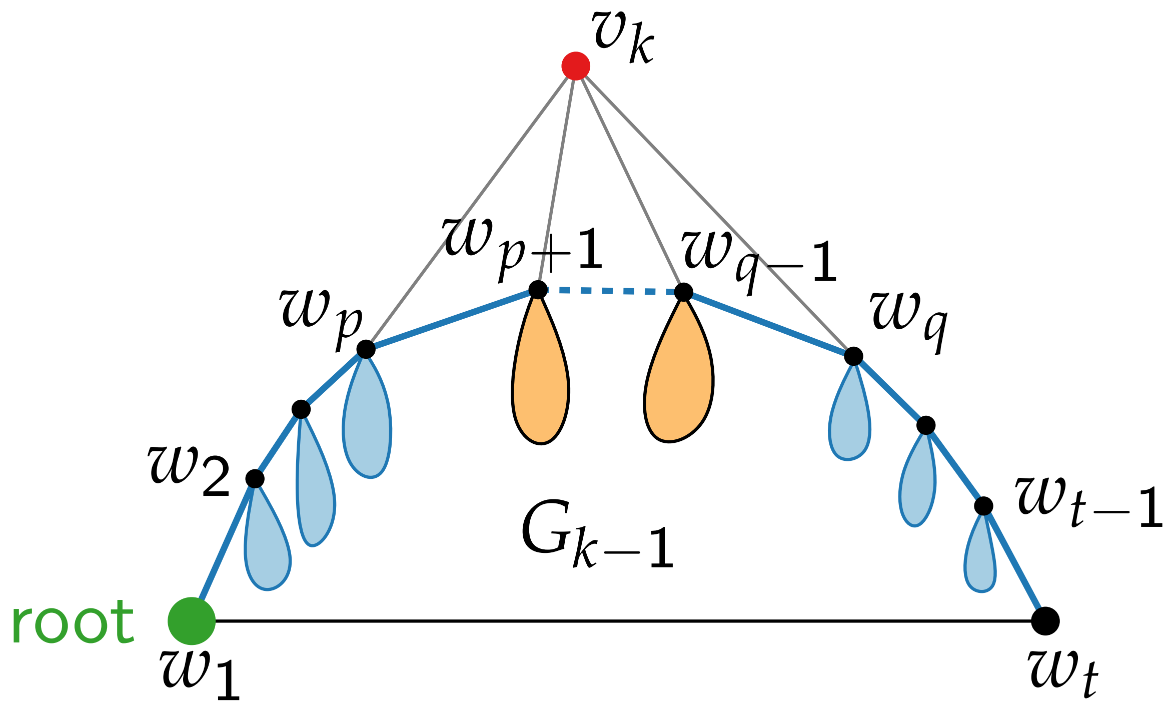


Shift method – linear time implementation

Relative x distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

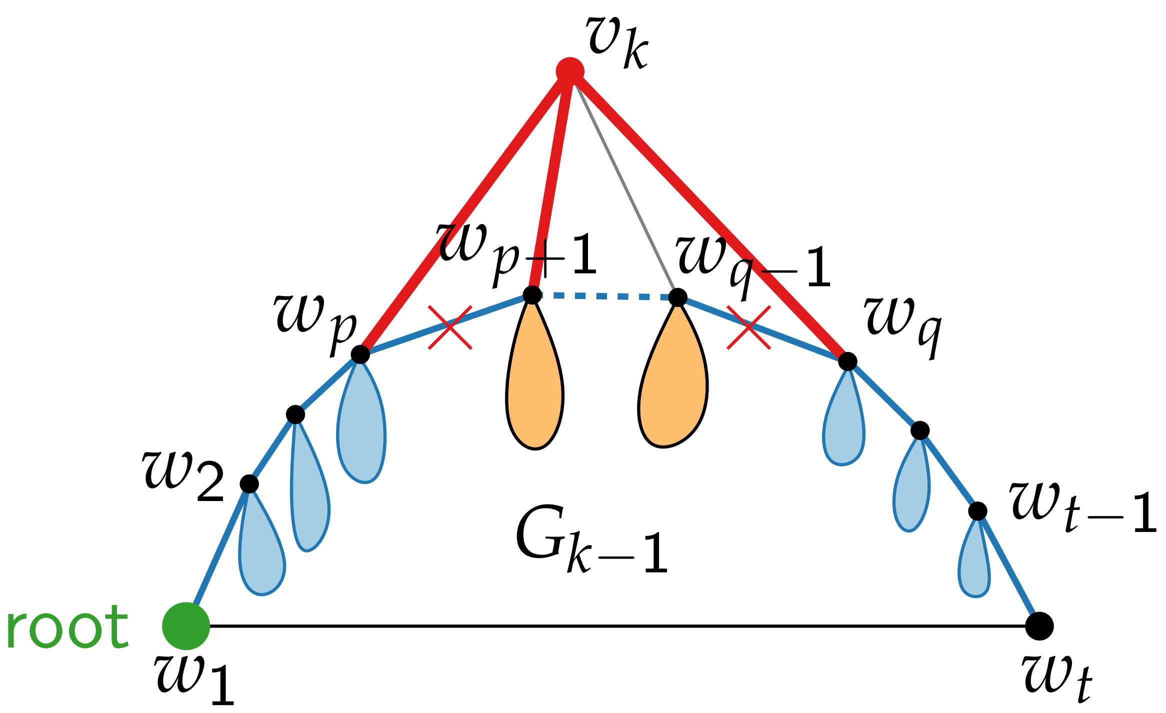


Shift method – linear time implementation

Relative x distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

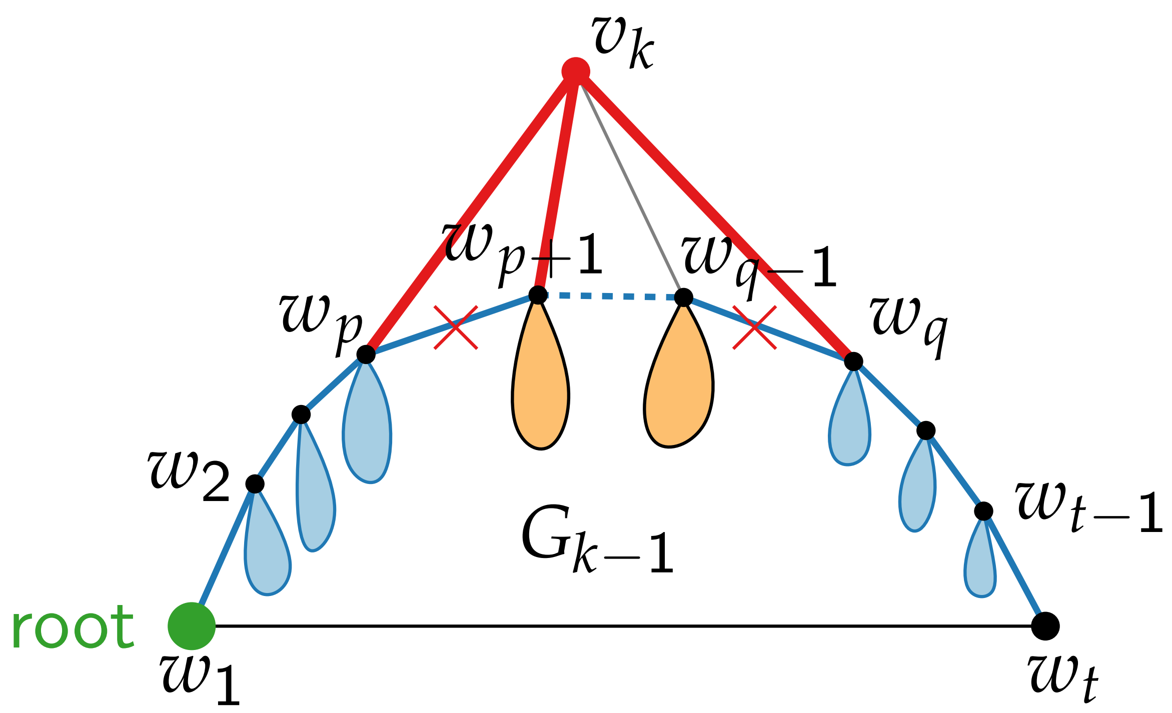


Shift method – linear time implementation

Relative x distance tree.

For each vertex v store
- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.
- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++

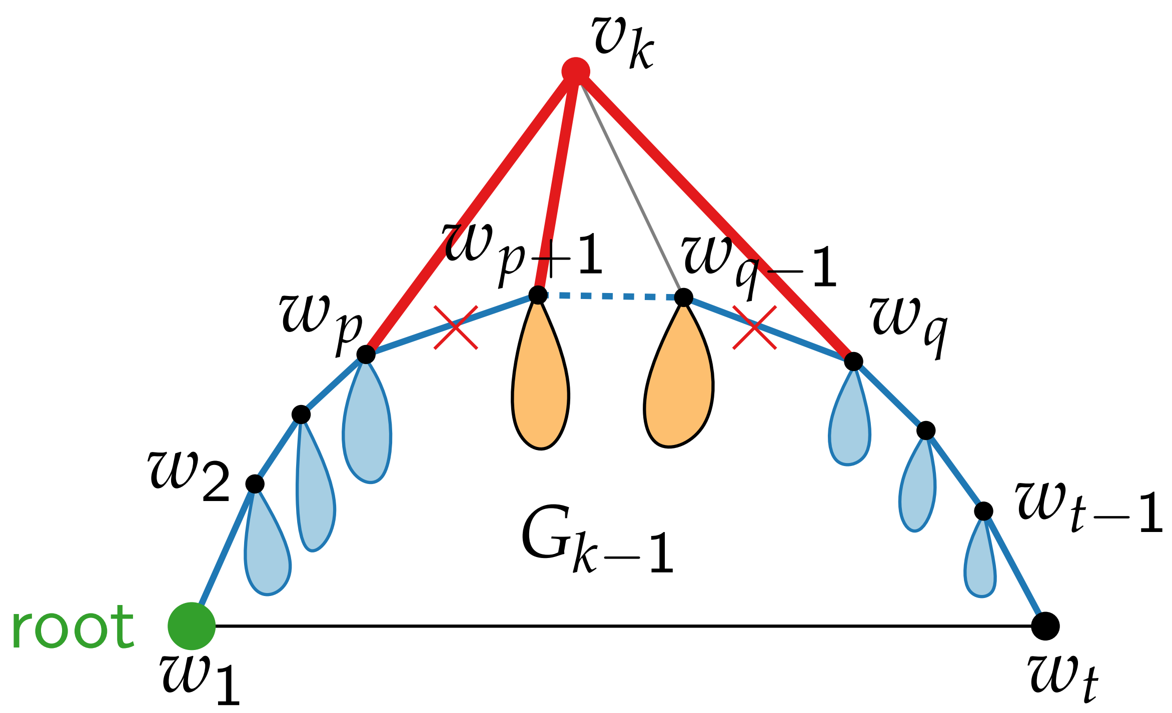


(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

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

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

Shift method – linear time implementation



Relative x distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)

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

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

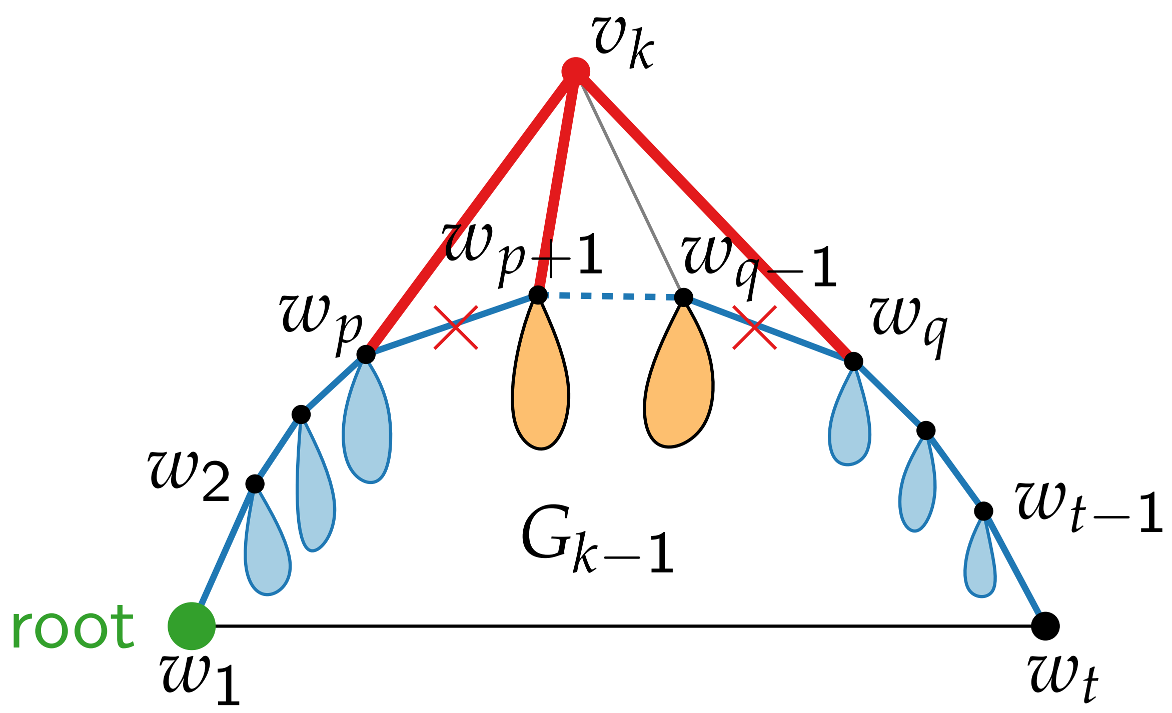
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

Relative x distance tree.

For each vertex v store
- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++



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

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

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

Shift method – linear time implementation

Relative x distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

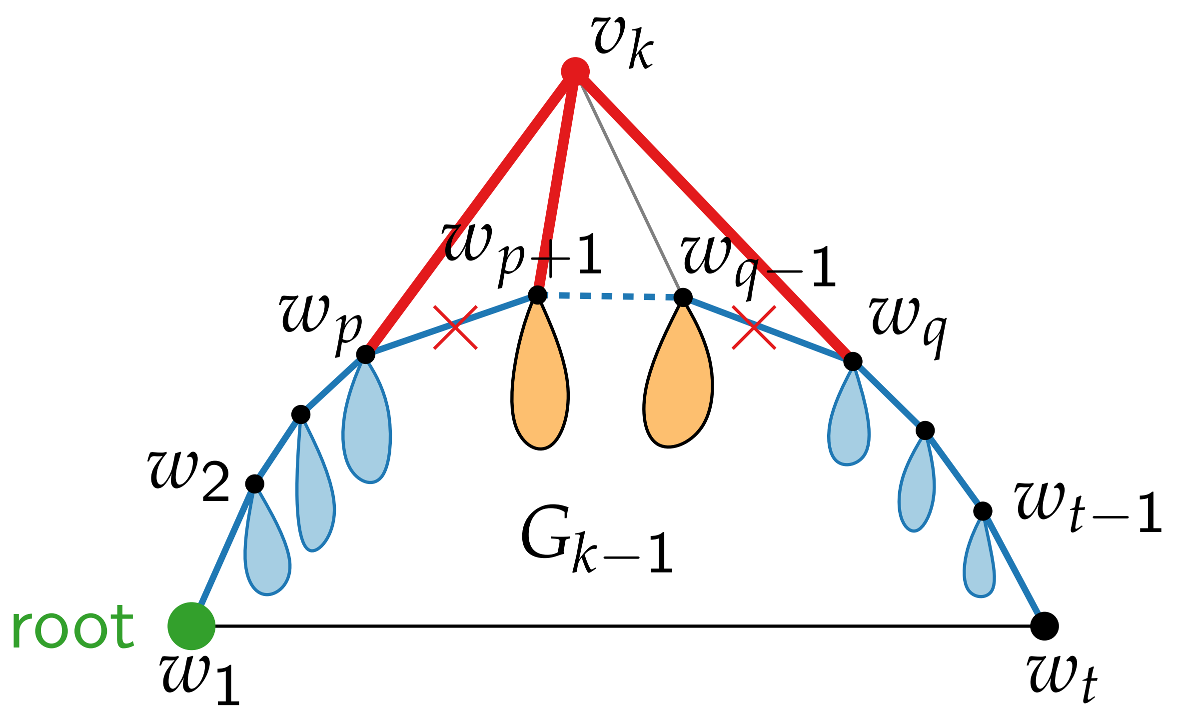
Calculations.

- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++

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

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

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



- After v_n, use preorder traversal to compute x-coordinates

Literature

- [PGD Ch. 4.2] for detailed explanation of shift method
- [dFPP90] de Fraysseix, Pach, Pollack "How to draw a planar graph on a grid" 1990 – original paper on shift method