Bitonic *st*-orderings of biconnected planar graphs

Martin Gronemann

Institut für Informatik
Universität zu Köln, Germany

September 24, 2014

Bitonic *st*-orderings of biconnected planar graphs

Martin Gronemann

Institut für Informatik
Universität zu Köln, Germany

September 24, 2014

Bitonic *st*-orderings of biconnected planar graphs

Martin Gronemann

Institut für Informatik
Universität zu Köln, Germany

September 24, 2014

Bitonic *st*-orderings
of biconnected planar graphs

Martin Gronemann

Institut für Informatik
Universität zu Köln, Germany

September 24, 2014

Introduction

Backstory

- Drawing undirected planar graphs

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)
- What about biconnected?

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)
- What about biconnected?

Solutions for triconnected $\Rightarrow$ biconnected

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)
- What about biconnected?

Solutions for triconnected $\Rightarrow$ biconnected

1. Augmentation to triconnected or maximal planar

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)
- What about biconnected?

Solutions for triconnected $\Rightarrow$ biconnected

1. Augmentation to triconnected or maximal planar
   ⚡ maximum degree related problems ⚡

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)
- What about biconnected?

Solutions for triconnected $\Rightarrow$ biconnected

1. Augmentation to triconnected or maximal planar
   ⚡ maximum degree related problems ⚡
2. Biconnected canonical & shelling ordering

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)
- What about biconnected?

Solutions for triconnected $\Rightarrow$ biconnected

1. Augmentation to triconnected or maximal planar
   ⚡ maximum degree related problems ⚡
2. Biconnected canonical & shelling ordering
   ⚡ not every internal node has a successor ⚡

Introduction

Backstory

- Drawing undirected planar graphs
- Canonical ordering for incremental drawing algorithms
- Limited to triconnected (Kant) or maximal planar (FPP)
- What about biconnected?

Solutions for triconnected $\Rightarrow$ biconnected

1. Augmentation to triconnected or maximal planar
   ⚡ maximum degree related problems ⚡
2. Biconnected canonical & shelling ordering
   ⚡ not every internal node has a successor ⚡
3. SPQR-tree approach

Biconnected planar graphs and *st*-orderings

What about *st*-orderings?

Biconnected planar graphs and *st*-orderings

What about *st*-orderings?

- Single source s and single sink t

Biconnected planar graphs and *st*-orderings

What about *st*-orderings?

- Single source s and single sink t
- Every $v \in V \setminus \{s, t\}$ has at least one predecessor and at least one successor

Biconnected planar graphs and *st*-orderings

What about *st*-orderings?

- Single source s and single sink t
- Every $v \in V \setminus \{s, t\}$ has at least one predecessor and at least one successor
- Works for (not necessarily planar) biconnected graphs

Biconnected planar graphs and *st*-orderings

What about *st*-orderings?

- Single source s and single sink t
- Every $v \in V \setminus \{s, t\}$ has at least one predecessor and at least one successor
- Works for (not necessarily planar) biconnected graphs
- Planar: s and t incident to the same face (here: $(s, t) \in E$)

Biconnected planar graphs and *st*-orderings

What about *st*-orderings?

- Single source s and single sink t
- Every $v \in V \setminus \{s, t\}$ has at least one predecessor and at least one successor
- Works for (not necessarily planar) biconnected graphs
- Planar: s and t incident to the same face (here: $(s, t) \in E$)

Biconnected planar graphs and *st*-orderings

What about *st*-orderings?

- Single source s and single sink t
- Every $v \in V \setminus \{s, t\}$ has at least one predecessor and at least one successor
- Works for (not necessarily planar) biconnected graphs
- Planar: s and t incident to the same face (here: $(s, t) \in E$)

Can we use *st*-orderings in the same way as canonical orderings?

Canonical vs. *st*-ordering

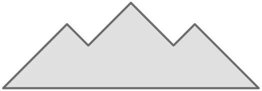
Experiment

"Let's use an st-ordering for the FPP-algorithm"

Canonical vs. *st*-ordering

Experiment

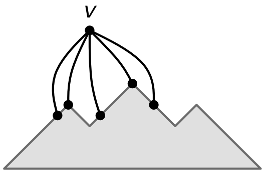
"Let's use an st-ordering for the FPP-algorithm"



Canonical vs. *st*-ordering

Experiment

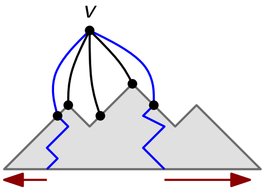
"Let's use an st-ordering for the FPP-algorithm"



Canonical vs. *st*-ordering

Experiment

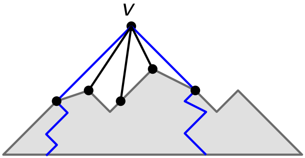
"Let's use an st-ordering for the FPP-algorithm"



Canonical vs. *st*-ordering

Experiment

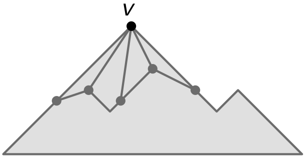
"Let's use an st-ordering for the FPP-algorithm"



Canonical vs. *st*-ordering

Experiment

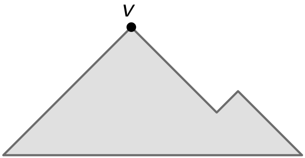
"Let's use an st-ordering for the FPP-algorithm"



Canonical vs. *st*-ordering

Experiment

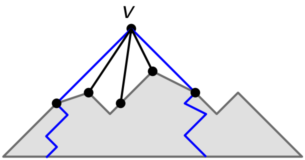
"Let's use an st-ordering for the FPP-algorithm"



Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"

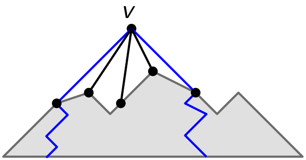


Multiple predecessors $\Rightarrow$ works!

Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"

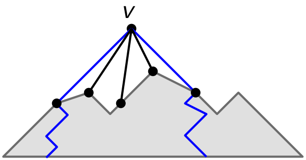


Multiple predecessors $\Rightarrow$ works!

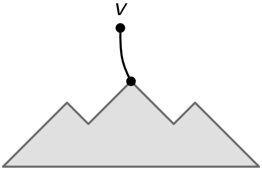
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

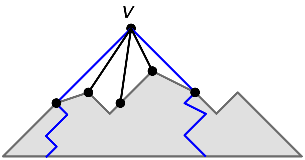


Single predecessor

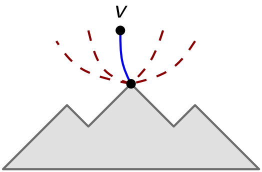
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

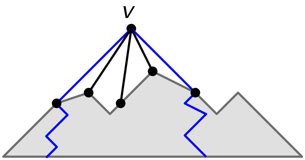


Single predecessor

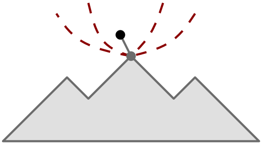
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

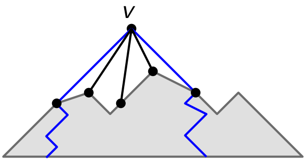


Single predecessor

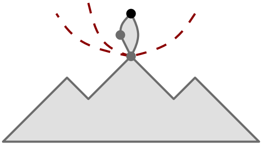
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

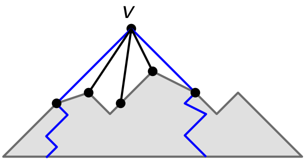


Single predecessor

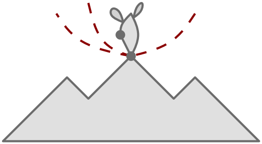
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

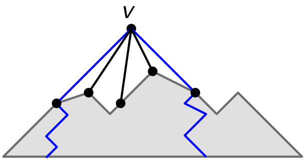


Single predecessor

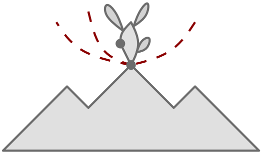
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors $\Rightarrow$ works!

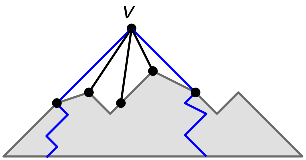


Single predecessor

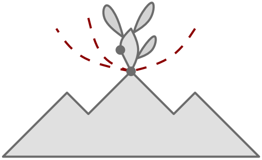
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

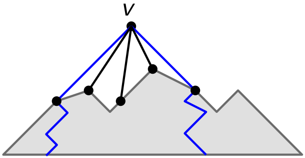


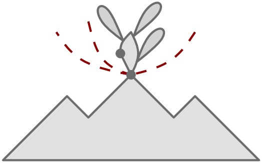
Single predecessor

Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"





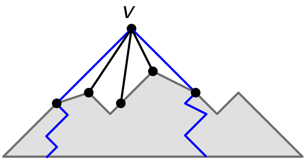
Multiple predecessors ⇒ works!

⚡ Single predecessor ⚡

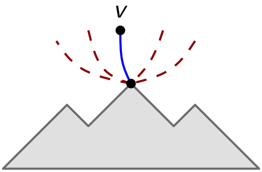
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

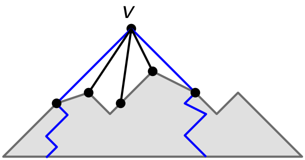


Single predecessor
(Harel & Sardas)

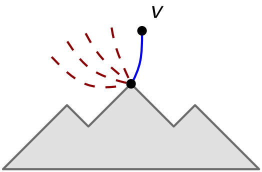
Canonical vs. *st*-ordering

Experiment

“Let’s use an st-ordering for the FPP-algorithm”



Multiple predecessors ⇒ works!

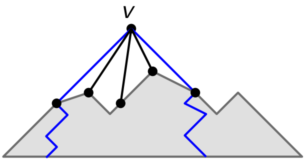


Single predecessor
(Harel & Sardas)

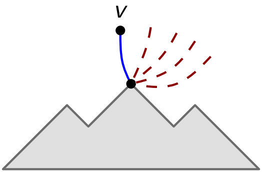
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors $\Rightarrow$ works!

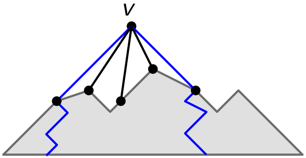


Single predecessor
(Harel & Sardas)

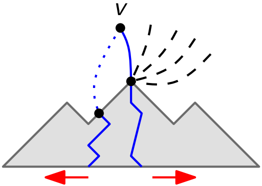
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors $\Rightarrow$ works!

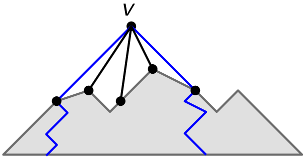


Single predecessor
(Harel & Sardas)

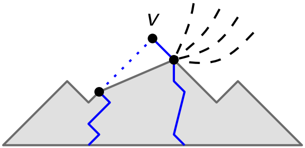
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

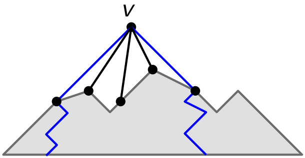


Single predecessor
(Harel & Sardas)

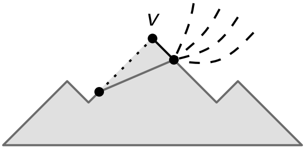
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

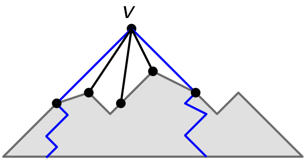


Single predecessor
(Harel & Sardas)

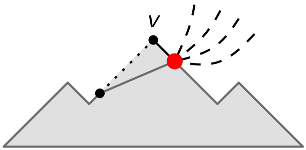
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!

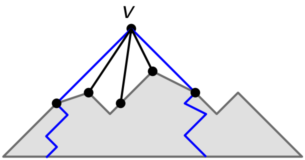


Single predecessor
(Harel & Sardas)

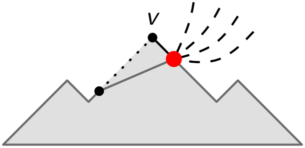
Canonical vs. *st*-ordering

Experiment

"Let's use an st-ordering for the FPP-algorithm"



Multiple predecessors ⇒ works!



Single predecessor
(Harel & Sardas)

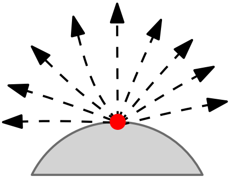
Observation

The unattached edges are the problem, not the single predecessor property.

Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.

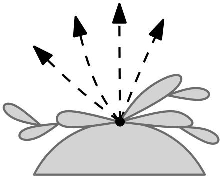


consecutive

Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.

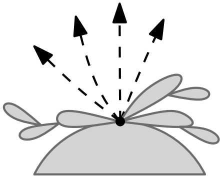


consecutive

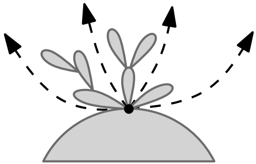
Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.



consecutive

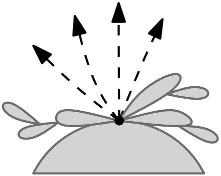


st-ordering

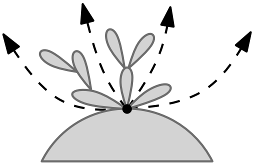
Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.



consecutive



st-ordering

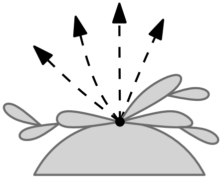
Question

Is there an *st*-ordering with this property?

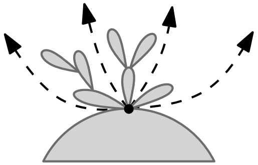
Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.



consecutive



st-ordering

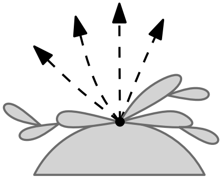
Question

Is there an *st*-ordering with this property? $\rightarrow$ **bitonic** ***st*-ordering**

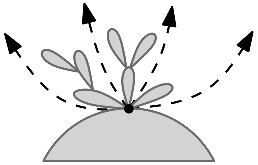
Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.



consecutive



st-ordering

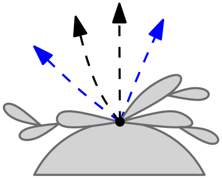
Question

Is there an *st*-ordering with this property? $\rightarrow$ **bitonic** ***st*****-ordering**

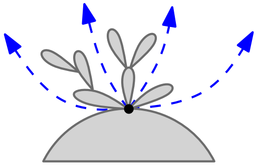
Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.



consecutive



st-ordering

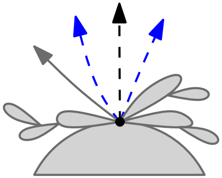
Question

Is there an *st*-ordering with this property? $\rightarrow$ **bitonic** ***st*-ordering**

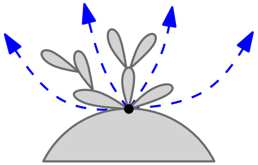
Canonical vs. *st*-ordering

Intuition

At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.



consecutive



st-ordering

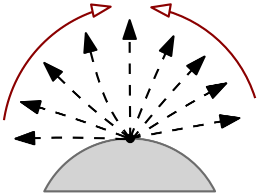
Question

Is there an *st*-ordering with this property? $\rightarrow$ **bitonic** ***st*-ordering**

Canonical vs. *st*-ordering

Intuition

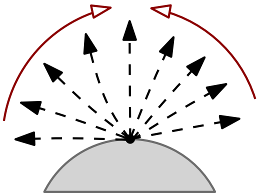
At any time, all incident edges that are **not yet present** in the current drawing, appear **consecutively** in the embedding.



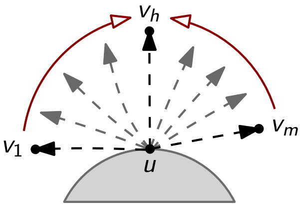
Question

Is there an *st*-ordering with this property? $\rightarrow$ **bitonic** ***st*-ordering**

Canonical vs. *st*-ordering



Canonical vs. *st*-ordering



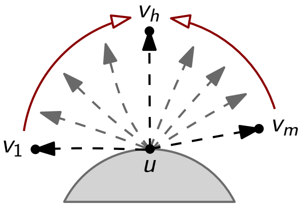
$$S(u) = \{v_1, \dots, v_m\}$$

Successors of u ordered as in the embedding

Canonical vs. *st*-ordering

Definition

Given an *st*-ordering $\pi : V \mapsto \{1, \ldots, |V|\}$ with $\pi(v)$ being the rank of $v \in V$ in the ordering.



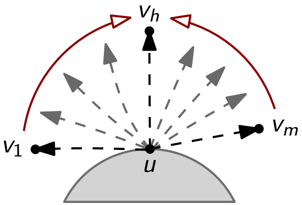
$$S(u) = \{v_1, \ldots, v_m\}$$

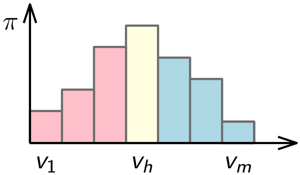
Successors of u ordered as in the embedding

Canonical vs. *st*-ordering

Definition

Given an *st*-ordering $\pi : V \mapsto \{1, \ldots, |V|\}$ with $\pi(v)$ being the rank of $v \in V$ in the ordering.





$$S(u) = \{v_1, \ldots, v_m\}$$

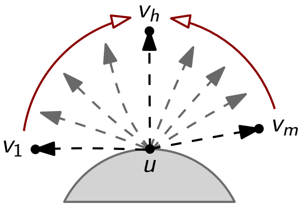
$$\pi(v_1) < \cdots < \pi(v_h) > \cdots > \pi(v_m)$$

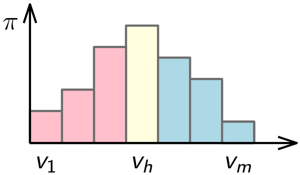
Successors of u ordered as in the embedding

Canonical vs. *st*-ordering

Definition

Given an *st*-ordering $\pi : V \mapsto \{1, \ldots, |V|\}$ with $\pi(v)$ being the rank of $v \in V$ in the ordering.





$$S(u) = \{v_1, \ldots, v_m\}$$

$$\pi(v_1) < \cdots < \pi(v_h) > \cdots > \pi(v_m)$$

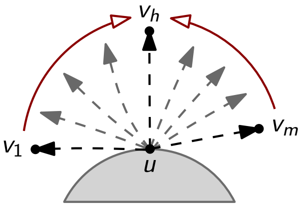
Successors of u ordered as in the embedding

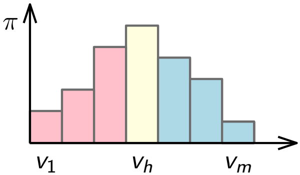
An increasing and then decreasing sequence $\Rightarrow$ **bitonic**

Canonical vs. *st*-ordering

Definition

Given an *st*-ordering $\pi : V \mapsto \{1, \ldots, |V|\}$ with $\pi(v)$ being the rank of $v \in V$ in the ordering.





$S(u) = \{v_1, \ldots, v_m\}$

Successors of u ordered as in the embedding

$\pi(v_1) < \cdots < \pi(v_h) > \cdots > \pi(v_m)$

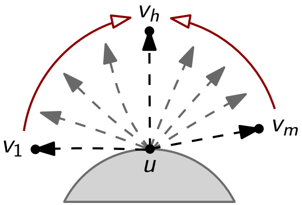
An increasing and then decreasing sequence $\Rightarrow$ **bitonic**

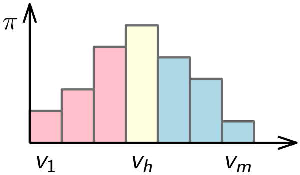
$S(u)$ is bitonic with respect to π

Canonical vs. *st*-ordering

Definition

Given an *st*-ordering $\pi : V \mapsto \{1, \ldots, |V|\}$ with $\pi(v)$ being the rank of $v \in V$ in the ordering.





$$S(u) = \{v_1, \ldots, v_m\}$$

Successors of u ordered as in the embedding

$$\pi(v_1) < \cdots < \pi(v_h) > \cdots > \pi(v_m)$$

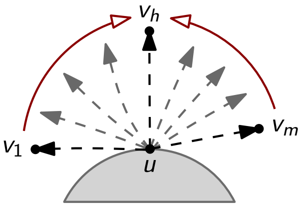
An increasing and then decreasing sequence $\Rightarrow$ **bitonic**

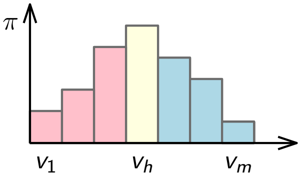
$\forall u \in V : S(u)$ is bitonic with respect to π

Canonical vs. *st*-ordering

Definition

Given an *st*-ordering $\pi : V \mapsto \{1, \ldots, |V|\}$ with $\pi(v)$ being the rank of $v \in V$ in the ordering.





$S(u) = \{v_1, \ldots, v_m\}$

Successors of u ordered as in the embedding

$\pi(v_1) < \cdots < \pi(v_h) > \cdots > \pi(v_m)$

An increasing and then decreasing sequence $\Rightarrow$ **bitonic**

$\forall u \in V : S(u)$ is bitonic with respect to $\pi \Rightarrow \pi$ is a **bitonic** *st*-order

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

bitonic *st*-ordering π

$+$

corresponding embedding

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

$+$

fixed embedding

?

bitonic *st*-ordering π

Bitonic *st*-ordering

Corollary

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

$+$

fixed embedding

counterexample

~~bitonic *st*-ordering π~~

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Sketch of the algorithm

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Sketch of the algorithm

- SPQR-tree approach to derive π

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Sketch of the algorithm

- SPQR-tree approach to derive π
- Canonical ordering for the R-nodes

Bitonic *st*-ordering

Theorem

Biconnected planar graph $G = (V, E)$

$+$

st-edge $(s, t) \in E$

$\Downarrow$ linear-time algorithm

bitonic *st*-ordering π

$+$

corresponding embedding

Sketch of the algorithm

- SPQR-tree approach to derive π
- Canonical ordering for the R-nodes
- P-nodes may require a change in the embedding

Bitonic *st*-ordering

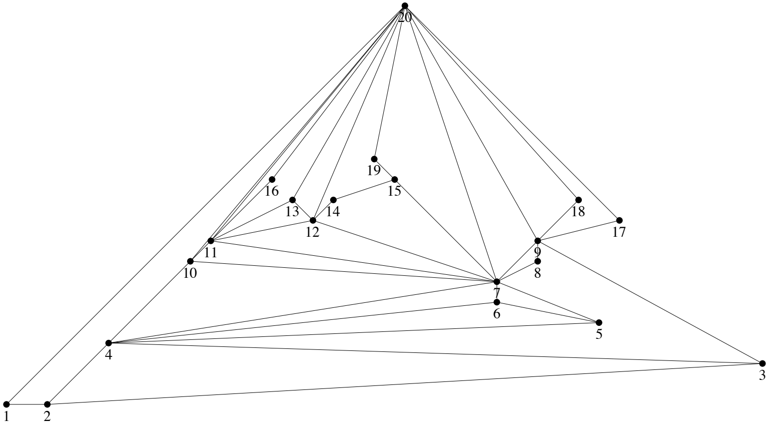
Experiment (revisited)

"Let's use a **bitonic** *st-ordering for the FPP-algorithm"*

Bitonic *st*-ordering

Experiment (revisited)

"Let's use a **bitonic** *st-ordering for the FPP-algorithm"*



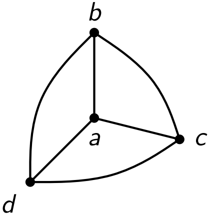
Rectilinear T-shaped contact representation

Task

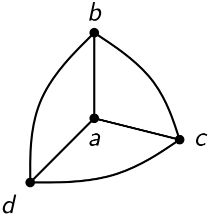
Rectilinear T-shaped contact representation

Task

- Given a (biconnected) planar graph

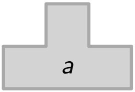


Rectilinear T-shaped contact representation



Task

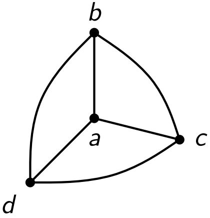
- Given a (biconnected) planar graph
- Vertices drawn as rectilinear T-shaped polygons

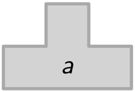


Rectilinear T-shaped contact representation

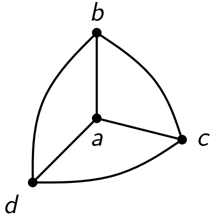
Task

- Given a (biconnected) planar graph
- Vertices drawn as rectilinear T-shaped polygons
- Here: upside down T's



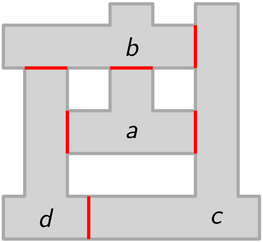


Rectilinear T-shaped contact representation



Task

- Given a (biconnected) planar graph
- Vertices drawn as rectilinear T-shaped polygons
- Here: upside down T's
- Adjacency expressed by touching sides of two polygons



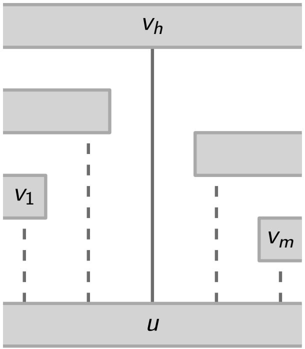
Rectilinear T-shaped contact representation

Idea

Rectilinear T-shaped contact representation

Idea

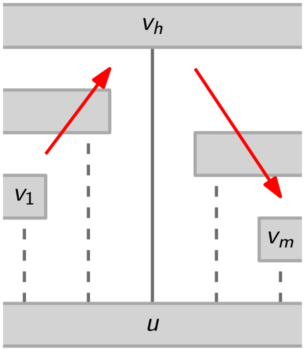
- Create a visibility representation, but with $y(v) = \pi(v)$



Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

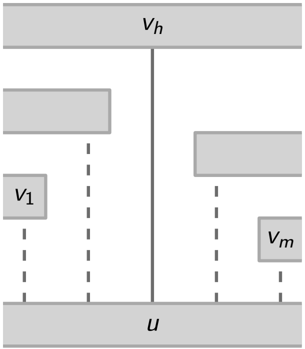


Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick



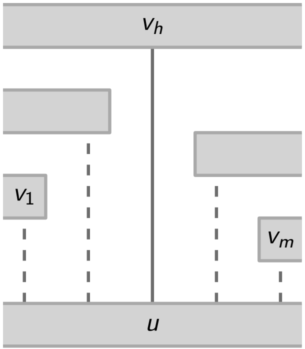
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

1. Grow a pole touching the highest successor from below



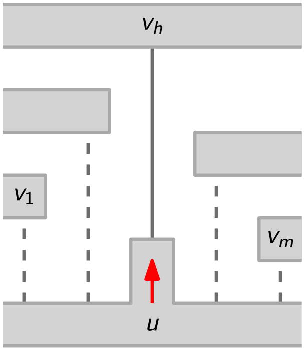
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

1. Grow a pole touching the highest successor from below



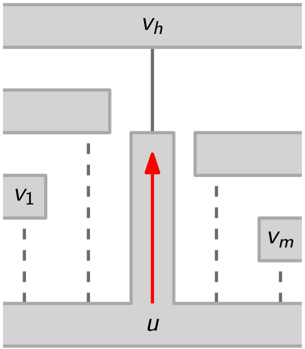
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

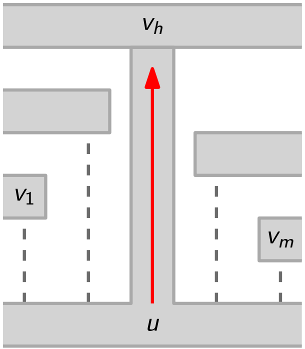
1. Grow a pole touching the highest successor from below



Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern



Simple trick

1. Grow a pole touching the highest successor from below

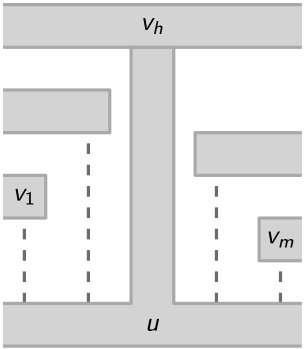
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

1. Grow a pole touching the highest successor from below



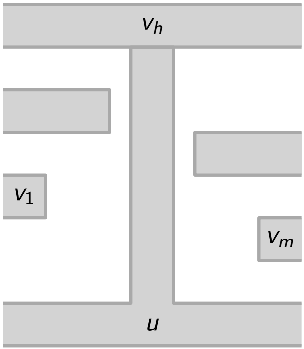
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

1. Grow a pole touching the highest successor from below



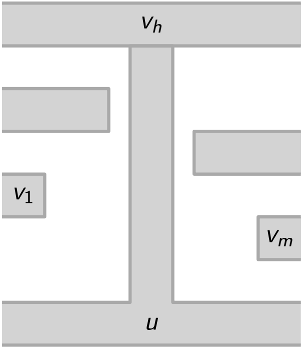
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

1. Grow a pole touching the highest successor from below
2. Pull the remaining ones towards it



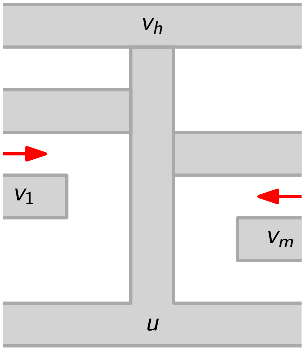
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

1. Grow a pole touching the highest successor from below
2. Pull the remaining ones towards it



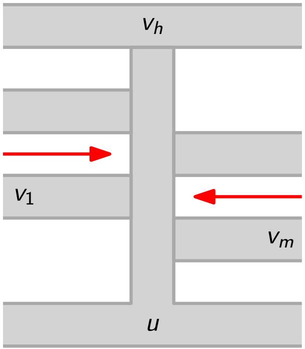
Rectilinear T-shaped contact representation

Idea

- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

1. Grow a pole touching the highest successor from below
2. Pull the remaining ones towards it



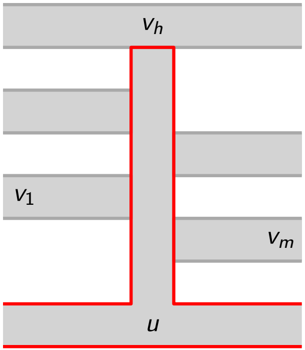
Rectilinear T-shaped contact representation

Idea

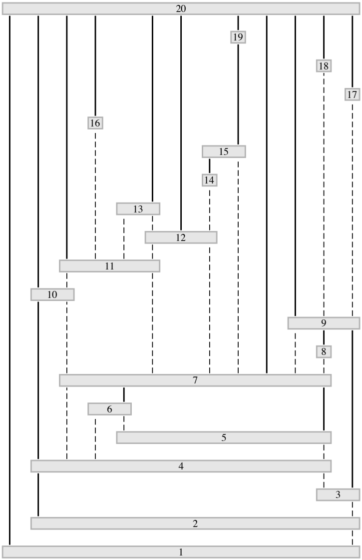
- Create a visibility representation, but with $y(v) = \pi(v)$
- Successors in an increasing and decreasing staircase pattern

Simple trick

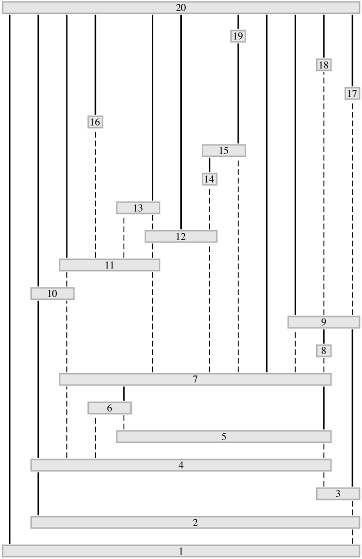
1. Grow a pole touching the highest successor from below
2. Pull the remaining ones towards it

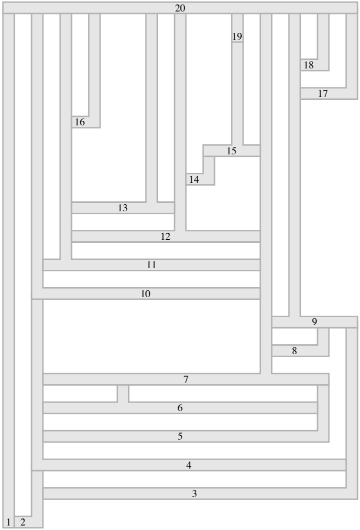


Rectilinear T-shaped contact representation



Rectilinear T-shaped contact representation





Conclusion

Bitonic *st*-ordering

- Special *st*-ordering obtainable in linear time
- Can be used similar to canonical orderings
- Requires a variable embedding setting
- Implementation in OGDF

Conclusion

Bitonic *st*-ordering

- Special *st*-ordering obtainable in linear time
- Can be used similar to canonical orderings
- Requires a variable embedding setting
- Implementation in OGDF

Ongoing and future work

- Directed graphs, esp. planar *st*-graphs
- Relation to upward straight-line drawings
- More applications, undirected and directed

Conclusion

Bitonic *st*-ordering

- Special *st*-ordering obtainable in linear time
- Can be used similar to canonical orderings
- Requires a variable embedding setting
- Implementation in OGDF

Ongoing and future work

- Directed graphs, esp. planar *st*-graphs
- Relation to upward straight-line drawings
- More applications, undirected and directed

Thank you for your attention!