Orthogonal Drawings of Graphs and Their Relatives
Part 1 - Topology-shape-metrics

Walter Didimo
University of Perugia
walter.didimo@unipg.it
Summary

• Part 1.1 – The topology-shape-metrics approach
• Part 1.2 – Engineering the topology-shape-metrics approach
• Part 1.3 – Ortho-polygon drawings
Part 1.1
The Topology-Shape-Metrics Approach
• Approach to compute an orthogonal drawing of a graph $G = (V, E)$
Topology-shape-metrics

**Input:** 4-graph \( G=(V,E) \)

\[ V = \{ u, v, w, k, p, q \} \]

\[ E = \{ (u, q), (u, v), (u, w), (v, q), (v, k), (v, w), (q, p), (q, k), (k, p), (k, w), (w, p) \} \]

**Output:** orthogonal drawing \( \Gamma \) of \( G \)
• **Topology (embedding):** set of (internal and external) faces, with possible crossing vertices
• **Shape (orthogonal representation):** vertex angles and edge bends
• **Metrics (orthogonal drawing):** vertex and bend coordinates

• These abstraction levels make it possible to design a drawing strategy in three phases:
  – planarization $\Rightarrow$ compute a topology (embedding)
  – orthogonalization $\Rightarrow$ compute a shape (orthogonal representation)
  – compaction $\Rightarrow$ compute a metrics (final drawing)
Topology-shape-metrics: Illustration

\[ V = \{1, 2, 3, 4, 5, 6\} \]
\[ E = \{(1,4), (1,5), (1,6), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6)\} \]
Planarization

**Objective:** Compute an embedding of G with **few crossings**

- **G planar** $\Rightarrow$ the planarization algorithm computes a planar embedding

- **G non-planar** $\Rightarrow$ the planarization algorithm computes an embedding with "small" number of crossings, i.e., an embedded planar graph G' obtained by replacing crossings with dummy vertices (**crossing vertices**)
Planarization: Crossing minimization

• Minimizing the number of edge crossings is NP-complete

• Determining the maximum planar subgraph is also NP-complete

• A simple planarization heuristic can work in two steps:
  – Step 1: compute a *maximal* planar embedded subgraph
  – Step 2: insert the remaining edges one by one trying to minimize the number of crossings
Planarization heuristic: Step 1

Input graph $G = (V, E)$
$V = \{1, 2, 3, 4, 5, 6\}$
$E = \{(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6)\}$

Maximal planar subgraph $G' = (V', E')$ of $G$
$V' = \{1, 2, 3, 4, 5, 6\}$
$E' = \{(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6)\}$

(1, 2) ⇒ planar
(1, 3) ⇒ planar
(1, 4) ⇒ planar
(1, 5) ⇒ planar
(1, 6) ⇒ planar
(2, 3) ⇒ planar
(2, 4) ⇒ planar
(2, 5) ⇒ planar
(2, 6) ⇒ planar
(3, 4) ⇒ planar
(3, 5) ⇒ non-planar
(4, 5) ⇒ planar
(4, 6) ⇒ non-planar
(5, 6) ⇒ planar
Planarization heuristic: Step 1

**maximal planar subgraph** $G' = (V', E')$ of $G$

$V' = \{1, 2, 3, 4, 5, 6\}$

$E' = \{(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4), (4, 5), (5, 6)\}$

non-planar edges of $G$: $(3, 5)$ e $(4, 6)$
Planarization heuristic: Step 2

Addition of edge (4,6)

- Shortest path on the dual graph of $G'$ between two faces incident to vertices 4 and 6, respectively.
- Insert a crossing vertex $x$ in $V'$ and update the dual graph of $G'$. 
Planarization heuristic: Step 2

addition of edge (3,5)

Shortest path on the dual graph of $G'$ between two faces incident to vertices 3 and 5, respectively

Insert a crossing vertex $w$ in $V'$
Planarization: Further references


• **C. Gutwenger, P. Mutzel, R. Weiskircher**: Inserting an Edge into a Planar Graph. Algorithmica 41(4): 289-308 (2005)

• **M. Chimani, C. Gutwenger**: Advances in the Planarization Method: Effective Multiple Edge Insertions. J. Graph Algorithms Appl. 16(3): 729-757 (2012)

Planarization: Open problem

• **Problem 1** Design planarization heuristics that compute embeddings with "few" crossings per edge

• **Remark**: Deciding whether a graph is k-planar (i.e., it has a drawing with at most k crossings per edge) is NP-hard
  – V. P. Korzhik and B. Mohar: Minimal obstructions for 1-immersions and hardness of 1-planarity testing. J. Graph Theory 72, 1 (2013)
Orthogonalization: Shape

- **Objective**: Compute a shape of G with *few bends*
  - *shape (orthogonal representation)*: described by the *angles at each vertex* and by the ordered *sequence of bends along each edge*
Orthogonalization: Bend minimization

• **Theorem** [Tamassia 1987] Given an embedded planar 4-graph \( G=(V,E) \), there exists a polynomial-time algorithm that computes an embedding preserving *orthogonal representation* of \( G \) with *minimum number of bends*.

• **Proof idea**
  - orthogonal representations of \( G \) \( \Leftrightarrow \) integer feasible flows in a suitable network \( N(G) \)
  - cost of the flow = number of bends of the orthogonal representation
  - computation of a bend-minimum orthogonal representation of \( G \) \( \Leftrightarrow \) computation of a min-cost flow in \( N(G) \)
\begin{flow network}
Flow network: Basic definitions

• flow network: directed graph \( N = (U, A) \)
  – every node \( v \in U \) is associated with an amount of flow \( b(v) \)
    • \( b(v) > 0 \) \( \Rightarrow \) \( v \) is a producer (it produces \( |b(v)| \) units of flow)
    • \( b(v) < 0 \) \( \Rightarrow \) \( v \) is a consumer (it consumes \( |b(v)| \) units of flow)
    • \( b(v) = 0 \) \( \Rightarrow \) \( v \) is a neutral node
  – it must be \( \sum_{v \in U} b(v) = 0 \)

– every arc \( e \in A \) is associated with three non-negative integers:
  • \( l(e) \) = lower capacity of \( e \)
  • \( u(e) \) = upper capacity of \( e \)
  • \( c(e) \) = cost of \( e \)
Flow network: Basic definitions

- **feasible flow** in $N$: a function $x: A \rightarrow \mathbb{N}$ such that:
  - $\forall e \in A \ l(e) \leq x(e) \leq u(e)$
  - $\forall v \in U \ \sum_{e \in \text{out}(v)} x(e) - \sum_{e \in \text{in}(v)} x(e) = b(v)$

- **cost of** $x$: $C(x) = \sum_{e \in A} c(e) \cdot x(e)$

- **min-cost flow** in $N$: feasible flow of minimum cost
\end{flow network}
Orthogonalization: Flow network – part I

- nodes of $N(G) \iff$ vertices and faces of $G$
- arc $(v, f)$ in $N(G) \iff$ angle at $v$ in face $f$
- flows on these arcs represent the values of the corresponding angles
- the flow originates from vertices (producers) and move towards faces (consumers)
Orthogonalization: Flow network – part I

- flow and angles
  - $k$ units of flow $\iff (k+1)90^\circ$ angle
  - a vertex $v$ produces $4\cdot\text{deg}(v)$ units of flow

```
• vertex of deg. 4 produces flow 0
• vertex of deg. 3 produces flow 1
• vertex of deg. 2 produces flow 2
• vertex of deg. 2 produces flow 2
• vertex of deg. 1 produces flow 3
```
Orthogonalization: Flow network – part I

- flow, angles, and face capacities
  - \( \text{cap}(f) \) = capacity of a face \( f \) \( \Leftrightarrow \) how many units of flow it can consume without generating bends on its boundary

![Flow network diagrams](attachment:image.png)

- face of deg 4 with 0 bends
- face of deg 4 with 1 bend
- face of deg 4 with 2 bends
- face of deg 5 with 0 bends
- face of deg 5 with 1 bend
Orthogonalization: Flow network – part I

- General rule for an *internal* face $f$
  - $\text{cap}(f) = \text{deg}(f) - 4$

- Implications:
  - if $f$ receives $k > \text{cap}(f)$ units of flow $\Rightarrow$ $f$ generates $k - \text{cap}(f)$ bends on its boundary, each forming a $90^\circ$ angle inside $f$
  - $\text{deg}(f) < 4 \Rightarrow \text{cap}(f)$ is negative $\Rightarrow f$ produces $(4 - \text{deg}(f))$ units of flow

![Diagram] (deg(f) = 3)  
![Diagram] (deg(f) = 2)
Orthogonalization: Flow network – part I

- for the external face \( h \): \( \text{cap}(h) = \text{deg}(h) + 4 \)
Orthogonalization: Flow network – part I

- flow, angles, and face capacities – summarizing
  - a vertex $v$ produces $4 - \deg(v)$ units of flow
  - an internal face $f$ of degree $> 3$ consumes $\deg(f) - 4$ units of flow
  - an internal face $f$ of degree $\leq 3$ produces $4 - \deg(f)$ units of flow
  - the external face $h$ consumes $\deg(h) + 4$ units of flow

![Diagram of a flow network with vertex labels and edge capacities]

- $l(e) = 0$
- $u(e) = 4 - \deg(v)$
- $c(e) = 0$
Orthogonalization: Flow network – part II

• How to model bends in the flow network? If a face $f$ receives more than $\text{cap}(f)$ units of flow, it must forward the excess to an adjacent face:
  – insert face-to-face arcs in $N(G)$ to allow flow exchange between adjacent faces
  – $k$ units of flow on an arc $(f, g)$ correspond to $k$ bends along an edge shared by $f$ and $g$; each bend forms an angle of $90^\circ$ inside $f$ and of $270^\circ$ inside $g$
  – face-to-face arcs have cost 1, so that the number of bends equals the total flow cost
Orthogonalization: Flow network – part II

- face-to-face arcs

\[ l(e) = 0 \]
\[ u(e) = +\infty \]
\[ c(e) = 1 \]
Orthogonalization: Flow network

- Flow network: **putting all together**
Orthogonalization: Flow network

- Final flow network $N(G)$
Orthogonalization: Flow and shape

- Example of flow and its corresponding shape
  - only arcs with non-zero flow are shown
Orthogonalization: Flow and shape

• Why an integer feasible flow always exists in N(G)

1) produced flow − consumed flow = 0

\[ \sum_{v \in V} (4 - \text{deg}(v)) + \sum_{f \text{ int:deg}(f) \leq 3} (4 - \text{deg}(f)) - \sum_{f \text{ int:deg}(f) > 3} (\text{deg}(f) - 4) - (\text{deg}(h) + 4) \right) = \]

\[ 4|V| - 2|E| - \sum_{f \in F} (\text{deg}(f) - 4) - 8 = \]

\[ 4 (|V| - |E| + |F| - 2) = 0 \quad (\text{by Euler’s formula}) \]

2) face-to-face arcs allow unbounded flow exchange
Orthogonalization: Computational cost

- Computing a min-cost flow of $O(n)$ given value in $N(G)$
  - $O(n^2 \log n)$ [Tamassia 1987]
  - $O(n^{7/4} \log n)$ [Garg and Tamassia 1996]
  - $O(n^{3/2})$ [Cornelsen and Karrenbauer 2011]

- Open Problem. Is there an $o(n^{3/2})$-time algorithm for the bend-minimization problem of plane 4-graphs?
Exercise (partial answer). Prove the following

**Theorem (unpublished).** Let $G$ be an embedded planar 4-graph with $n$ vertices and all internal faces of degree less than 5. There exists an $O(n)$-time algorithm that computes an embedding-preserving bend-minimum orthogonal representation of $G$. 
Orthogonalization: Solution
Orthogonalization: Solution
Orthogonalization: Solution
Orthogonalization: Solution
Orthogonalization: Solution

run a BFS visit from the external face
Orthogonalization: Solution

#bends = \( \sum (sp(\bigcirc) - 1) \) + \( \sum 2(sp(\bigcirc) - 1) \) + \( \sum sp(\square) = 1 + 2 + 3 = 6 \)
Compaction

- **Objective**: Assign vertex and bend coordinates such that the final drawing has either small area or small total edge length
  - for some orthogonal representations it is impossible to minimize both these parameters together

- For the given drawings:
  - Minimum area: Area = 36, tel = 48
  - Minimum total edge length: Area = 42, tel = 47
Compaction: Complexity

• Minimizing the area (or the total edge length) of an orthogonal representation is NP-hard

• The problem is polynomial-time solvable if all faces are rectangles
  – this result is generalized to a larger class of orthogonal representations called turn-regular (see later)
Compaction: General strategy

1. Transform the shape into a rectangular shape
   a) replace every bend with a dummy vertex
   b) add dummy edges and vertices until all faces are rectangles
2. Compute vertex coordinates
3. Remove all dummy edges and vertices
Compaction: Step 1

a) replace every bend with a dummy vertex
Compaction: Step 1

b) add dummy edges and vertices until all faces are rectangles

split recursively each *internal face* every time a subsequence RRL is found while walking *clockwise*
Compaction: Step 1

b) add dummy edges and vertices until all faces are rectangles

split recursively each *internal face* every time a subsequence *RRL* is found while walking *clockwise*
Compaction: Step 1

b) add dummy edges and vertices until all faces are rectangles

... a more complex example
b) add dummy edges and vertices until all faces are rectangles

... a more complex example
Compaction: Step 1

b) add dummy edges and vertices until all faces are rectangles

split recursively the *external face*

every time a subsequence *LRL* or *LRR* is found while walking

*counterclockwise*
Compaction: Step 1

b) add dummy edges and vertices until all faces are rectangles

split recursively the external face every time a subsequence LRL or LRR is found while walking counterclockwise
Compaction: Step 2

• Compute vertex coordinates

- assign the x-coordinates so that the width is minimized
- assign the y-coordinates so that the height is minimized
- for a rectangular shape this leads to minimum area
Compaction: Step 2

- Compute vertex coordinates
- Find the x-coordinates so that the width is minimized
  - create super-nodes that group the vertices in the same vertical chain
  - connect two super-nodes with a left-to-right directed edge if the corresponding chains are connected in the shape
  - assign to chains the x-coordinates computed by an optimal topological numbering of their super-nodes
Compaction: Step 2

- Compute vertex coordinates

- Find the y-coordinates so that the height is minimized
  - uses a super-node that groups the vertices in the same horizontal chain
  - connect two super-nodes with a bottom-to-top directed edge if the corresponding chains are connected in the shape
  - assign to chains the y-coordinates computed by an *optimal topological numbering* of their super-nodes
Compaction: Step 3

- Remove dummy edges and vertices

The described algorithm works in $O(n)$ time
Compaction: Total edge length

For a rectangular shape of given width and height, it is possible to minimize the total edge length within its dimensions in polynomial time.
Compaction: Total edge length

- use two flow networks, one for the vertical compaction ($N_{\text{ver}}$) and the other for the horizontal compaction ($N_{\text{hor}}$)
The flow on each arc corresponds to the length of the corresponding edge of the orthogonal shape.
Compaction: Total edge length

the fact that the node of the network associated with each internal face is a neutral node guarantees the consistency of the face dimensions
Compaction: Further issues

**Observation:** The dummy vertices and edges added in the rectangularization phase represent a constraint, which may strongly affect the result of the compaction algorithm.

Example 1
Compaction: Further issues

**Observation:** The dummy vertices and edges added in the rectangularization phase represent a constraint, which may strongly affect the result of the compaction algorithm.

Example 2
A planar orthogonal representation is turn-regular if it has no pairs of kitty-corners (opposing reflex vertices) inside a face.
Compaction: Turn-regularity

Two orthogonal representations of the same plane graph

*not* turn-regular

turn-regular
Compaction: Turn-regularity

**Theorem.** Let $H$ be an orthogonal representation of an embedded planar 4-graph with $n$ vertices. It is possible to test in $O(n)$ time whether $H$ is turn-regular. In the positive case, an orthogonal drawing of $H$ of minimum area can be computed in $O(n)$ time.

Part 1.2
Engineering the Topology-Shape-Metrics Approach
Practical considerations

• Real graphs typically contain high-degree vertices (with degree larger than 4)

• Many applications usually need to customize a generic drawing algorithm by imposing some drawing constraints
  – vertices represented as boxes of prescribed sizes
  – specific edges that cannot cross or that cannot bend
  – ...

• In the following we briefly discuss the above issues
High-degree vertices: First strategy

• After the planarization step, replace each high-degree vertex with a **dummy face**, having all vertices of degree 3

• Apply the topology-shape-metrics approach with some constraints that guarantee that each dummy face is drawn as a **rectangle**

• In the final drawing **dummy faces will be shown as boxes**
High-degree vertices: First strategy

- After the planarization step, replace each high-degree vertex with a **dummy face**, having all vertices of degree 3.
High-degree vertices: First strategy

- Apply the topology-shape-metrics approach with some constraints that guarantee that each dummy face is drawn as a rectangle.
High-degree vertices: First strategy

- constraints on the orthogonalization algorithm

- each edge of the dummy face boundary is forced to be straight

- this is done by deleting the face-to-face arcs incident to the dummy face node in the flow network
High-degree vertices: First strategy

- In the final drawing, dummy faces will be shown as boxes.
Drawbacks of this strategy

• No control on the dimensions of high-degree vertices
  – the corresponding dummy faces may be stretched a lot in the compaction phase
• Real-world applications may require all vertices of the same dimensions
High-degree vertices: Second strategy

- Use a different model with all vertices of the same size (Kandinsky)
  - Fößmeier and Kaufmann: Drawing high degree graphs with low bend numbers, Graph Drawing (1995)
High-degree vertices: Kandinsky

1. introduction of angles of $0^\circ$
2. each face has an area strictly greater than 0
High-degree vertices: Kandinsky

- Unfortunately, minimizing the number of bends in the Kandinsky model is NP-complete:
  - T. Bläsius, G. Brückner, I. Rutter: Complexity of Higher-Degree Orthogonal Graph Embedding in the Kandinsky Model. ESA (2014)

- But the problem is polynomial-time solvable with few additional restrictions (simple Kandinsky)
1. there cannot be two edges incident to the same side of a vertex if there is at least one unused side of the vertex

2. If there are multiple edges incident to the same side of a vertex, all of them except the first (in clockwise order) must bend in the same direction (e.g. to the right)
High-degree vertices: simple Kandinsky

- To compute a bend-minimum orthogonal representation in the simple Kandinsky model extend Tamassia's flow network
  - each high-degree vertex $v$ becomes a consumer instead of a producer; it consumes flow $\deg(v) - 4$, received by its incident faces

face-to-vertex arcs

- $l(e) = 0$
- $u(e) = 1$
- $c(e) = 1$
High-degree vertices: simple Kandinsky

- Interpretation of the flow on the new kind of arcs
  - one unit of flow on an arc \((f,v)\) represents an angle of 0° and causes 1 bend
High-degree vertices: simple Kandinsky

- *Compaction* of simple Kandinsky
  - reduced to the compaction algorithm for classical orthogonal shapes
Handling constraints

- The topology-shape-metrics approach makes it possible to deal with several types of constraints in each phase:
  - topology constraints
  - shape constraints
  - metrics constraints
Topology constraints

- Some topology constraints
  - edges that cannot cross (uncrossable edges)
  - subsets of vertices that must lie on the same face boundary
  - groups of edges that must be consecutive around a common end-vertex

- Handled in the planarization phase
Topology constraints

- Some **topology constraints**
  - edges that cannot cross (uncrossable edges)

To make an edge *uncrossable*, the planarization algorithm is modified by removing the corresponding edge in the dual graph; *a shortest path in the dual cannot cross the primal edge*
Topology constraints

• Some **topology constraints**
  – subsets of vertices that must lie on the same face boundary
  
  the planarization algorithm is applied after the insertion of a
  \textit{“star-gadget” of uncrossable edges}

![Planarization algorithm](image)
Topology constraints

- Some **topology constraints**
  - groups of edges that must be consecutive around a common end-vertex

the planarization algorithm is applied after the insertion of a suitable
“star-gadget” for each group
Topology constraints

• Other topology constraints
Shape constraints

- Some shape constraints
  - deciding the number of bends on an edge (e.g., no bend or any number or a specific number)
  - deciding the turn direction of an edge (left or right)
  - bounding or fixing the values of vertex angles

- Handled in the orthogonalization phase by suitably modifying capacities and/or costs of the arcs of the flow network
Some shape constraints

1. deciding the number of bends on an edge (e.g., no bend or any number or a specific number)
2. deciding the turn direction of an edge (left or right)
3. bounding or fixing the values of vertex angles

Modify the cost of the face-to-face arcs or fix the flow
Delete one of the two face-to-face arcs
Change lower and upper capacities of the vertex-to-face arc
Metrics constraints

• Some *metrics constraints*
  − deciding vertex dimensions (width and the height of each single vertex)
  − deciding the attaching point of each edge

• Handled in the compaction phase
Metrics constraints

• Some metrics constraints
  – deciding vertex dimensions (width and height of each single vertex)

• Idea
  – start from a drawing of a succinct Kandinsky shape
  – expand vertices iteratively, by inserting extra rows and columns in the drawing, according to the desired vertex dimensions (expressed in terms of grid units)
  – compact the drawing again
  – uncompress edges to get the final drawing
Metrics constraints

- start from a drawing of a succinct Kandinsky shape
Metrics constraints

- expand vertices iteratively, by inserting extra rows and columns in the drawing, according to the desired vertex dimensions (expressed in terms of grid units)

width = 2
height = 1
Metrics constraints

- compact the drawing again
  - replace each box with a “suitable” number of vertices of zero dimension (points)
  - replace each bend with a dummy vertex
Metrics constraints

- compact the drawing again
  - create a **dummy cage** that includes the drawing and divide it into **horizontal strips** (extra dummy vertices and segments are created)
Metrics constraints

- compact the drawing again
  - compact horizontally by computing a min-cost-flow in a suitable network

- flow represents edge lengths;
- produced flow = width of the cage
- arcs associated with box-vertex segments have fixed flow value (lower cap. = upper capacity)
- arcs associated with dummy segments have cost 0
Metrics constraints

- compact the drawing again
  - do the same to compact vertically – and repeat until no improvement happens
- decompress edges to get the final drawing
Further references

• M. Eiglsperger, U. Fößmeier, M. Kaufmann: Orthogonal graph drawing with constraints. SODA 2000: 3-11

• M. Eiglsperger, M. Kaufmann: Fast Compaction for Orthogonal Drawings with Vertices of Prescribed Size. Graph Drawing 2001: 124-138
Implementations

• Some graph drawing libraries that implement the topology-shape-metrics approach or other orthogonal drawing algorithms:
  – Tom Sawyer Software (www.tomsawyer.com/)
Applications: Hermes

Applications: DBDraw

Applications: WhatsOnWeb (WOW)

Applications: Hybrid visualizations

Applications: MatchOMan (MOM)

Part 1.3
Ortho-polygon Drawings
From edge complexity to vertex complexity

• If vertices are drawn as polygons, one may save edge bends

rectangle visibility representation

From edge complexity to vertex complexity

- It can be tested in polynomial time if an embedded graph admits a rectangle visibility representation
Ortho-polygon drawings

- Generalization of rectangle visibility representations – a vertex can be an ortho-polygon with both convex and reflex corners

vertex-complexity = maximum number of reflex corners in a vertex
Ortho-polygon drawings: Existence

• Not all embedded graphs admit an ortho-polygon drawing

• Necessity:
  – the embedded graph is biplanar, i.e., the edge set can be partitioned into two planar subsets (e.g., vertical and horizontal in the drawing)
Ortho-polygon drawings: Existence

• Biplanarity is not sufficient

this face is not realizable, because it should have more than 4 convex corners and no reflex corners
Ortho-polygon drawings: Existence

• Not all embedded graphs admit an ortho-polygon drawing

• Necessity:
  – the embedded graph is biplanar, i.e., the edge set can be partitioned into two planar subsets (e.g., vertical and horizontal in the drawing)
  – each face with only crossing-vertices has degree four
Ortho-polygon drawings: Existence

• Questions:
  – can we test whether an embedded graph admits an ortho-polygon drawing?
  – can we compute (if any) an ortho-polygon drawing with minimum vertex complexity? (i.e., minimum number of reflex corners per vertex)
  – if yes, can we also minimize the total number of reflex corners within the minimum vertex complexity?
Ortho-polygon drawings: Existence

• Questions:
  – can we test whether an embedded graph admits an ortho-polygon drawing?
  – can we compute (if any) an ortho-polygon drawing with minimum vertex complexity? (i.e., minimum number of reflex corners per vertex)
  – if yes, can we also minimize the total number of reflex corners within the minimum vertex complexity?

• Answer: yes, by using a variant of Tamassia's flow network we can solve everything in polynomial time
Ortho-polygon drawings: Expansion graph
Ortho-polygon drawings: Characterization

Ortho-polygon drawing of $G$

Orthogonal drawing of $G^*$
Ortho-polygon drawings: Characterization

- P1. each red vertex has a 180° angle inside its node-face
- P2. each real edge has no bend
Ortho-polygon drawings: Flow network

- **P1.** each red vertex has a 180° angle inside its node-face
- **P2.** each real edge has no bend

Test and computation of an ortho-polygon drawing, with minimum number of reflex corners in total.
Ortho-polygon drawings: Flow network

- P1. each red vertex has a 180° angle inside its node-face
- P2. each real edge has no bend

test and computation of an ortho-polygon drawing, with minimum number of reflex corners in total and at most h reflex corners per face
Ortho-polygon drawings: Flow network

Each of the four units of flow corresponding to a convex corner in f will traverse a node-face at most once.

\[ f \leq 4n \]

Apply a binary search within \([0, 4n]\) for the determining the best value for \(h\).
Ortho-polygon drawings: Flow network

Computational complexity
- flow network size $= O(n)$
- flow value $= O(n)$
- flow cost $\chi = O(n^2)$

Min-cost flow algorithm time for fixed $h$: $O(\chi^{3/4} n \log^{1/2} n) = O(n^{5/2} \log^{1/2} n)$

Min-cost flow algorithm time $\times$ binary-search time ($O(\log n)$): $O(n^{5/2} \log^{3/2} n)$
Ortho-polygon drawings: 1-plane graphs

• Remarks:
  – every 1-plane graph admits an ortho-polygon drawing:
    • 2-connected 1-plane graphs may require vertex complexity \( \Omega(n) \)
    • 3-connected 1-plane graphs may require vertex complexity 2
    • 3-connected 1-plane graphs always admit an ortho-polygon drawing with vertex complexity at most 5 [G. Liotta, F. Montecchiani, A. Tappini: Ortho-Polygon Visibility Representations of 3-Connected 1-Plane Graphs. Graph Drawing 2018: 524-537]
Ortho-polygon drawings: Example

2-connected 1-plane graph with vertex complexity 3
Ortho-polygon drawings: Open problems

- **Problem 1.** Reduce the time-complexity of computing ortho-polygon drawings of minimum vertex complexity on general graphs.

- **Problem 2.** Reduce the theoretical gap between upper bound (5) and lower bound (2) on the vertex complexity of ortho-polygon drawings of 3-connected 1-planar graphs.
Ortho-polygon drawings: Experiments

(a) Running time.

(b) % of vertices with complexity $i$ (VC-$i$-V%).

(c) Running time.

(d) % of vertices with complexity $i$ (VC-$i$-V%).