Jack Spalding-Jamieson (Jack S-J)
jacksj@uwaterloo.ca

Cheriton School of Computer Science
University of Waterloo

UNIVERSITY OF WATERLOO

Graph Morphing via Orthogonal Box Drawings

Joint work with my supervisors Therese Biedl and Anna Lubiw

Presentation as part of MMath degree, 2023

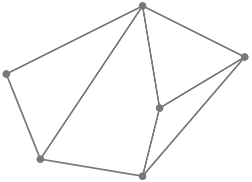
Graphs, Graph Drawings, Grids

Graph: vertex set V, edge set $E \subset \binom{V}{2}$ (no multiedges, no self-loops)

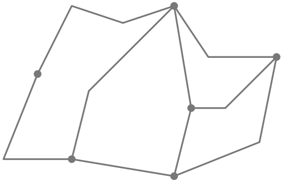
Graphs, Graph Drawings, Grids

Graph: vertex set V, edge set $E \subset \binom{V}{2}$ (no multiedges, no self-loops)

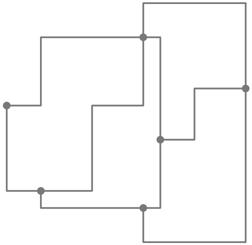
Graph drawing: Representation of a graph on a plane
Planar graph drawing: Graph drawing where edges do not intersect



Vertices: Points
Edges: Line segments



Vertices: Points
Edges: Polylines

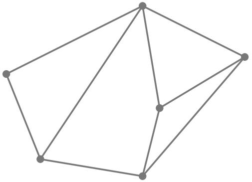


Vertices: Points
Edges: Orthogonal polylines

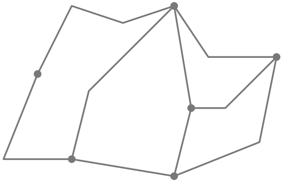
Graphs, Graph Drawings, Grids

Graph: vertex set V, edge set $E \subset \binom{V}{2}$ (no multiedges, no self-loops)

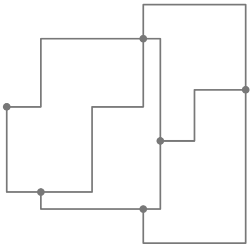
Graph drawing: Representation of a graph on a plane
Planar graph drawing: Graph drawing where edges do not intersect



Vertices: Points
Edges: Line segments



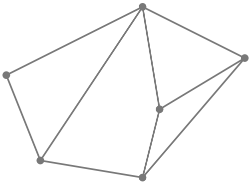
Vertices: Points
Edges: Polylines



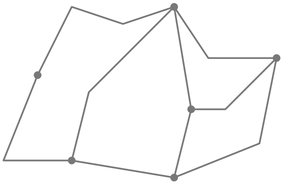
Vertices: Points
Edges: Orthogonal polylines

Our graphs will always be planar graphs (planar drawing exists).

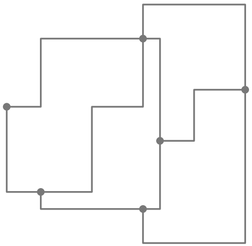
Point Drawings



Planar straight-line drawing



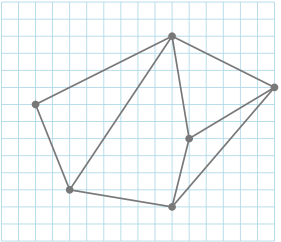
Planar poly-line drawing



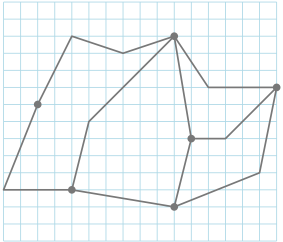
Planar orthogonal point drawing

Vertices are points.

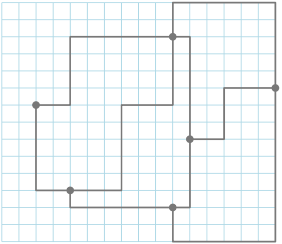
Point Drawings



Planar straight-line drawing



Planar poly-line drawing

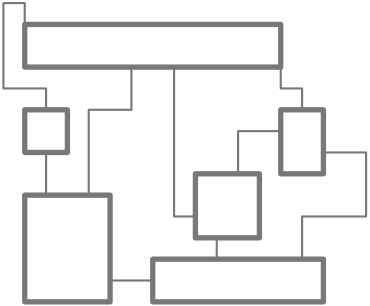


Planar orthogonal point drawing

Vertices are points.

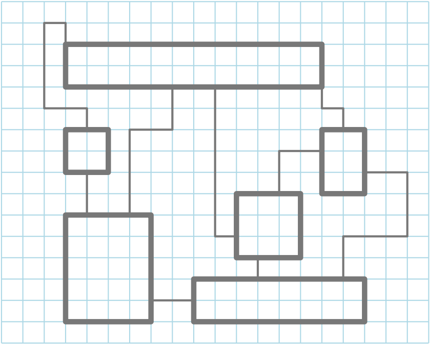
Can be drawn on a grid.

Non-Point Drawings



Planar orthogonal box drawing
Vertices: Axis-aligned rectangles
Edges: Orthogonal polylines

Non-Point Drawings

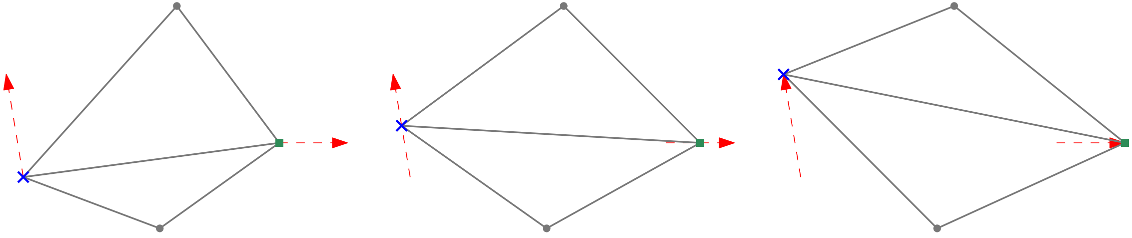


Planar orthogonal box drawing
Vertices: Axis-aligned rectangles
Edges: Orthogonal polylines

Can be drawn on a grid.

Graph Morphing

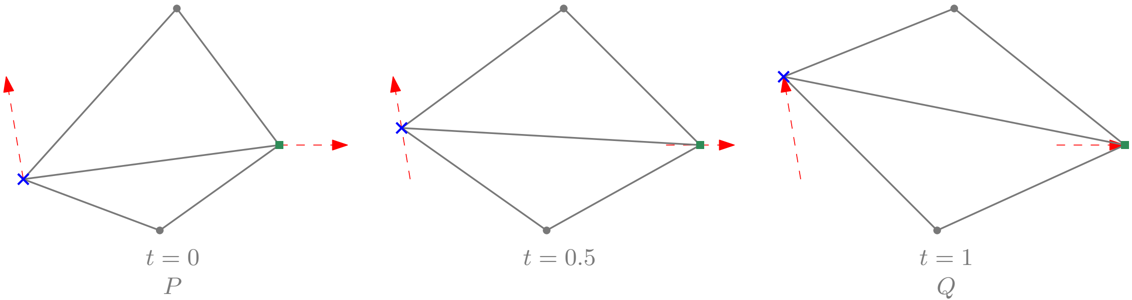
Morph: Continuously deform between drawings



$t = 0$ P $t = 0.5$ $t = 1$ Q

Graph Morphing

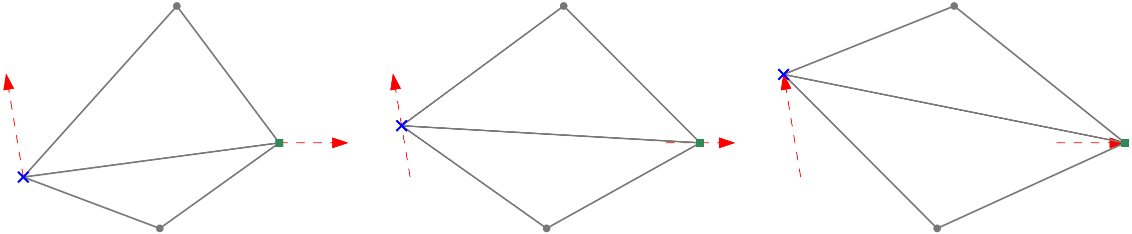
Morph: Continuously deform between drawings



Linear morph: Linearly interpolate vertex (and other) locations

Graph Morphing

Morph: Continuously deform between drawings



$t = 0$
P

$t = 0.5$

$t = 1$
Q

Linear morph: Linearly interpolate vertex (and other) locations

Morphs are always reversible!

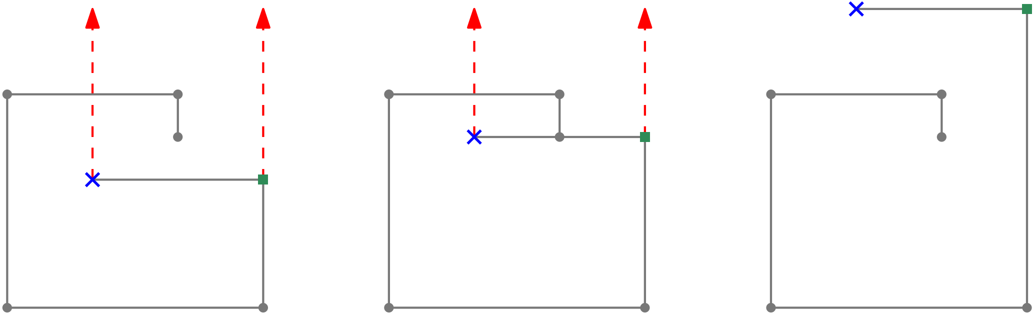
Planarity-Preserving Morphs

Planarity-Preserving Morph: At all times t, the "interpolated" drawing is also planar.

Planarity-Preserving Morphs

Planarity-Preserving Morph: At all times t, the "interpolated" drawing is also planar.

Non-planarity-preserving morph:

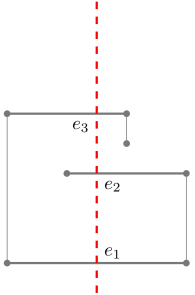


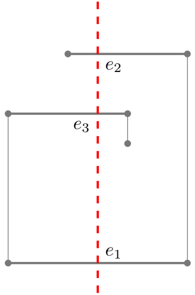
$t = 0$ $t = \frac{1}{4}$ $t = 1$

Unidirectional Linear Morphs

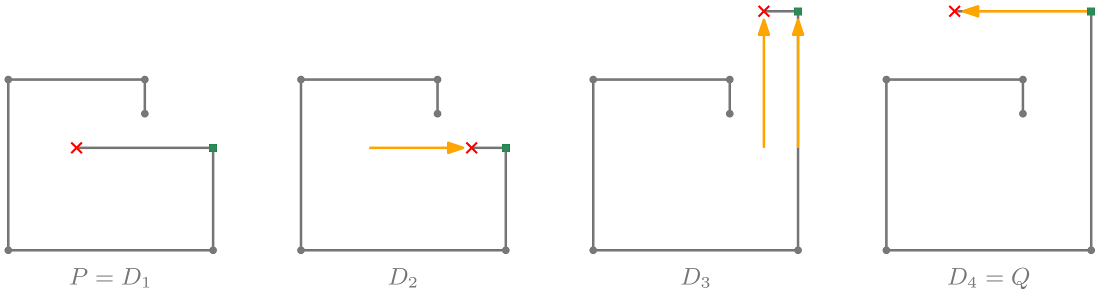
Unidirectional linear morph: All movement directions are parallel.

Fact (Alamdari et al., Kleist et al.): Unidirectional linear morphs are planarity-preserving if and only if every line parallel to the direction of movement has the same intersection order in both drawings.



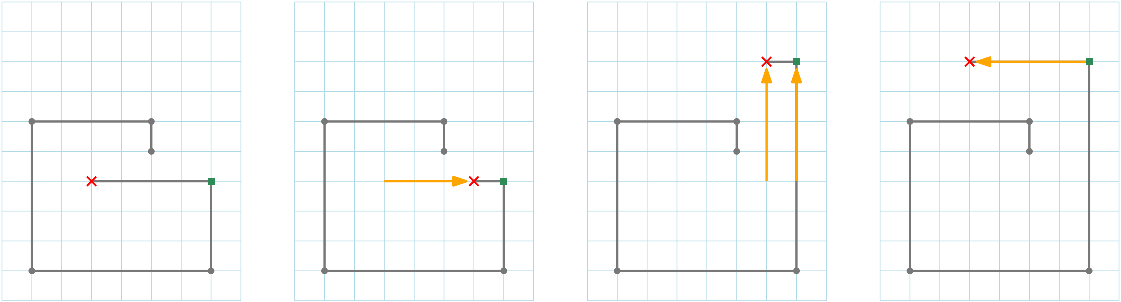


Linear Morphs Sequences



The above are the **explicit intermediate drawings**.
Entire morph is represented by the sequence D_1, D_2, D_3, D_4.

Linear Morph Sequences on a Grid



$P = D_1$ D_2 D_3 $D_4 = Q$

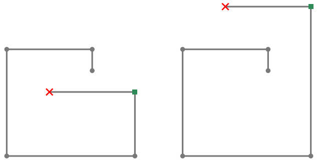
Explicit drawings are on a grid.
Implicit (interpolated) drawings are not.

Linear Morphs Sequences that Add/Remove Bends

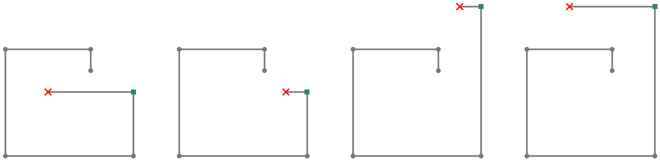
Degenerate bend: Bend that "isn't used" (coincident or 180° angle).
Equivalent drawings: Drawings that differ only by degenerate bends.

The Linear Morph Problem

Input:
'Compatible' pair of drawings (labelled)



Output:
Planarity-preserving linear morph sequence (list of drawings)



Objectives: Numerous!

Previous Results

Input: ('Compatible') pair of drawings
Output: Planarity-preserving linear morph sequence

	Graph/Drawing Class	Num linear morphs	Grid-size side-length	Bends per edge	Comput. Model	Time Complexity
Alamdari et al. (2017)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^3)$
Klemz (2021)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^2 \log n)$

Previous Results

Input: ('Compatible') pair of drawings
Output: Planarity-preserving linear morph sequence

	Graph/Drawing Class	Num linear morphs	Grid-size side-length	Bends per edge	Comput. Model	Time Complexity
Alamdari et al. (2017)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^3)$
Klemz (2021)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^2 \log n)$
Klemz (2021)	2-connected	$O(n)$	Expo.	0	Powerful	$O(n^2)$

Previous Results

Input: ('Compatible') pair of drawings
Output: Planarity-preserving linear morph sequence

	Graph/Drawing Class	Num linear morphs	Grid-size side-length	Bends per edge	Comput. Model	Time Complexity
Alamdari et al. (2017)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^3)$
Klemz (2021)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^2 \log n)$
Klemz (2021)	2-connected	$O(n)$	Expo.	0	Powerful	$O(n^2)$
Lubiw & Petrick (2011)	Straight-line	$O(n^6)$	$O(n^3)$	$O(n^5)$	Word RAM	Polynomial

Previous Results

Input: ('Compatible') pair of drawings
Output: Planarity-preserving linear morph sequence

	Graph/Drawing Class	Num linear morphs	Grid-size side-length	Bends per edge	Comput. Model	Time Complexity
Alamdari et al. (2017)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^3)$
Klemz (2021)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^2 \log n)$
Klemz (2021)	2-connected	$O(n)$	Expo.	0	Powerful	$O(n^2)$
Lubiw & Petrick (2011)	Straight-line	$O(n^6)$	$O(n^3)$	$O(n^5)$	Word RAM	Polynomial
This work (main result)	Connected	$O(n)$	$O(n)$	$O(1)$	Word RAM	$O(n^2)$

Previous Results

Input: ('Compatible') pair of drawings
Output: Planarity-preserving linear morph sequence

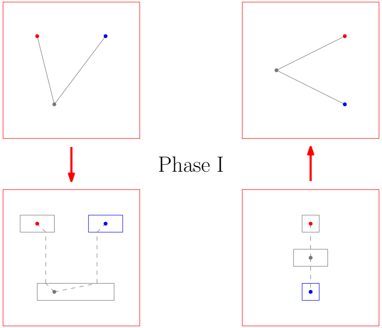
	Graph/Drawing Class	Num linear morphs	Grid-size side-length	Bends per edge	Comput. Model	Time Complexity
Alamdari et al. (2017)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^3)$
Klemz (2021)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^2 \log n)$
Klemz (2021)	2-connected	$O(n)$	Expo.	0	Powerful	$O(n^2)$
Lubiw & Petrick (2011)	Straight-line	$O(n^6)$	$O(n^3)$	$O(n^5)$	Word RAM	Polynomial
This work (main result)	Connected	$O(n)$	$O(n)$	$O(1)$	Word RAM	$O(n^2)$
Biedl et al. (2013)	Connected Orthogonal	$O(n^2)$	$O(n)$	$O(n)$	Word RAM	Polynomial
Van Goethem et al. (2022)	Orthogonal	$O(n)$	Polynomial	$O(1)$	Word RAM	Polynomial
This work (main method)	Connected Ortho-Box	$O(n)$	$O(n)$	$O(1)$	Word RAM	$O(n^2)$

Grid size assumes input fits on the same grid.

Above table is not comprehensive.

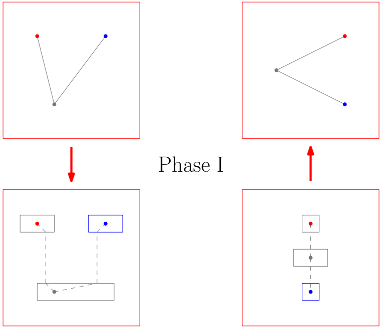
High-Level Overview

First: Reduce straight-line drawing morphing problem → orthogonal box drawing morphing problem.

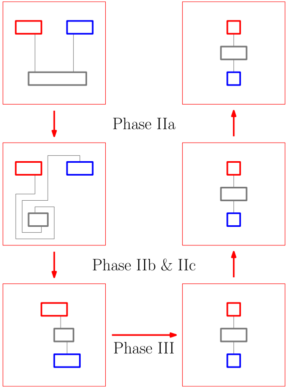


High-Level Overview

First: Reduce straight-line drawing morphing problem → orthogonal box drawing morphing problem.

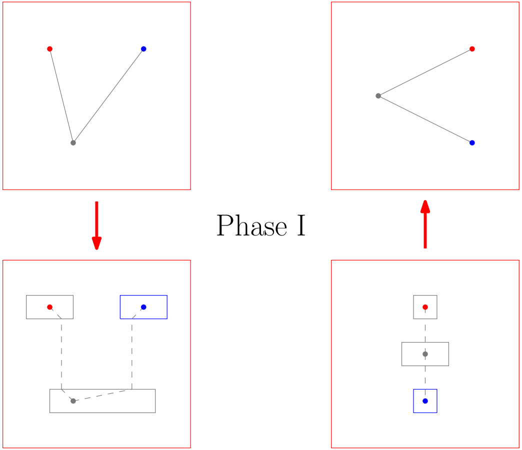


Second: Solve orthogonal box drawing morphing problem using (improved) techniques for orthogonal point drawing morphing problem.



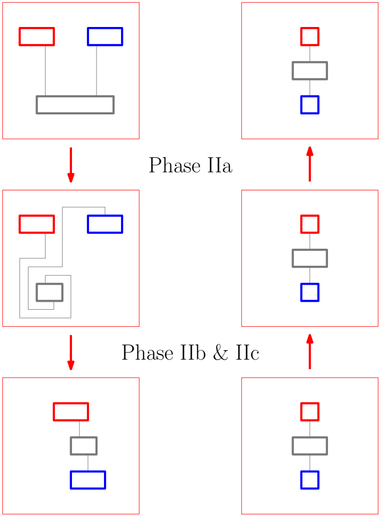
Phase I

High-level: Reduce to box drawing morphs.
Need to do a morph, and give a reduction.



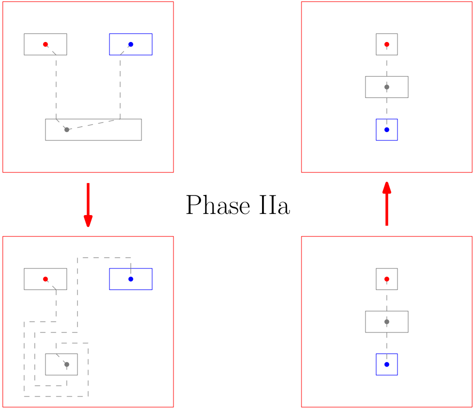
Phase II

High-level: Reduce to parallel box drawing morphs (only lengths differ). Only need to morph (not a different drawing type).



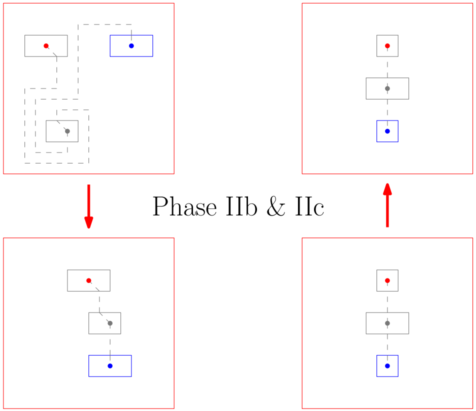
Phase IIa

High-level: Move ports. Add bends to do so.



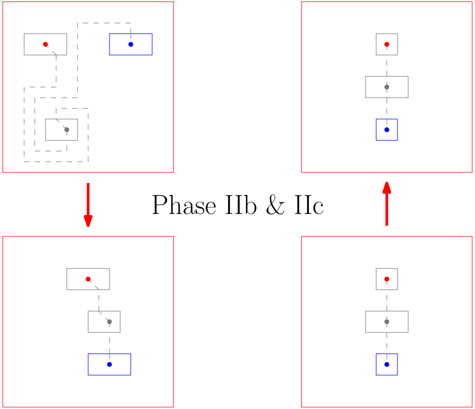
Phase IIb

High-level: Do some (global) analysis on the edges.



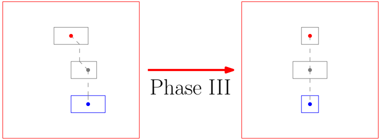
Phase IIc

High-level: Use analysis to get rid of bends.

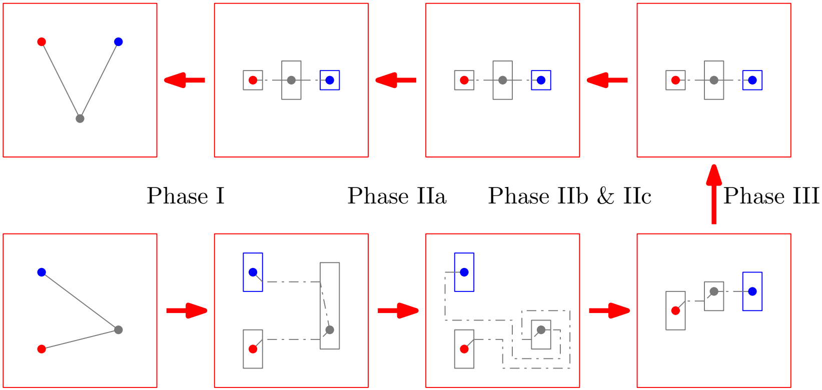


Phase III

High-level: Use black-box result to morph parallel orthogonal box drawings (i.e., adjust lengths).



The Phases—All Together



An example of all phases on a very simple drawing.

- Phase I: Reduce to boxes.
- Phase IIa: Edit ports.
- Phase IIb: Analyze edges.
- Phase IIc: Globally 'unify'.
- Phase III: Morph unified drawings.

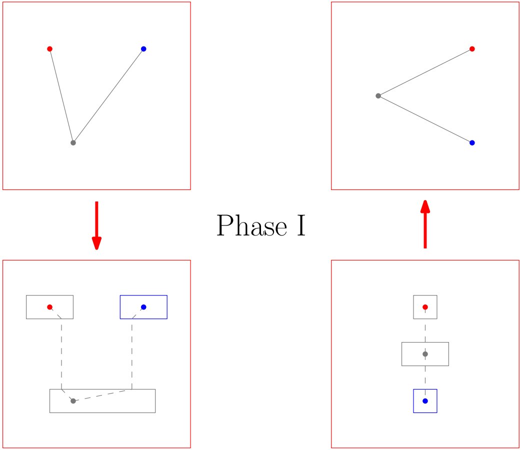
And now, details!

Phase I

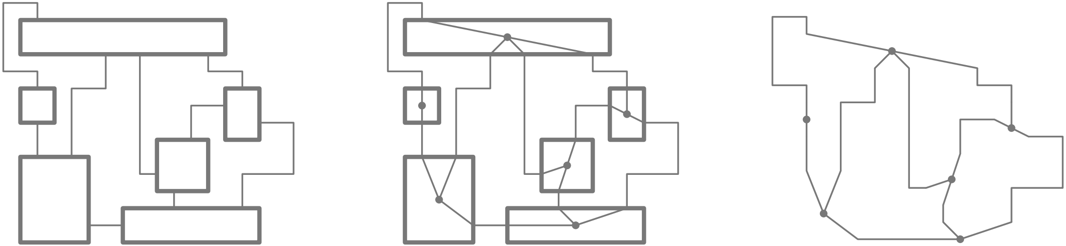
High-level: Reduce to box drawing morphs.
Need to do a morph, and give a reduction.

Phase I Overview

- **Input:** Straight-line drawing pair
- **Output:** Box drawing pair
- Also need a reduction (morphing box drawings ≊ morphing point drawings).

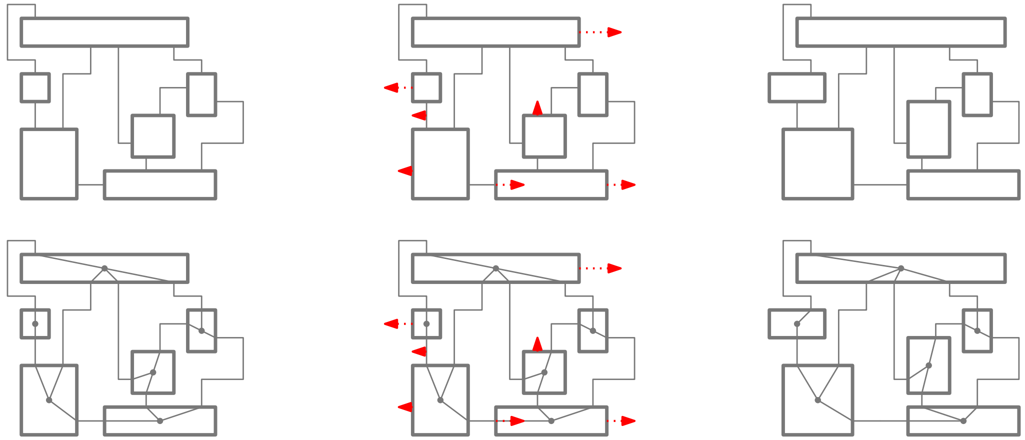


Reduction: Admitted Drawings (1)



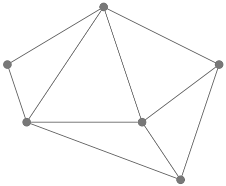
Orthogonal box drawing | Both | Admitted poly-line drawing

Reduction: Admitted Drawings (2)

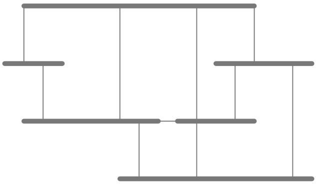


Morph of orthogonal box drawings $\Longrightarrow$ morph of admitted drawings

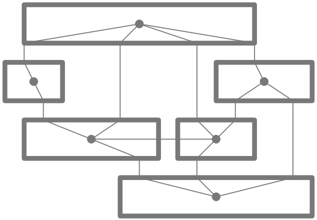
Computing Box Drawings: Visibility Representations as an Intermediary



A planar straight-line drawing P.

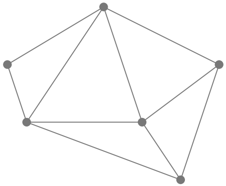


A visibility representation that can be computed from P.

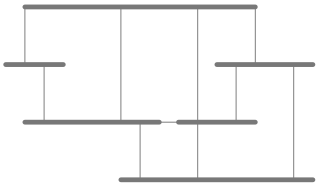


An orthogonal box drawing, and corresponding admitted drawing P', which can both be computed from P.

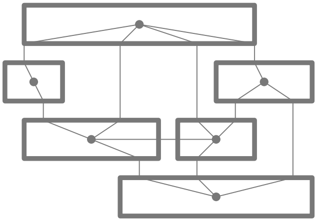
Computing Box Drawings: Visibility Representations as an Intermediary



A planar straight-line drawing P.



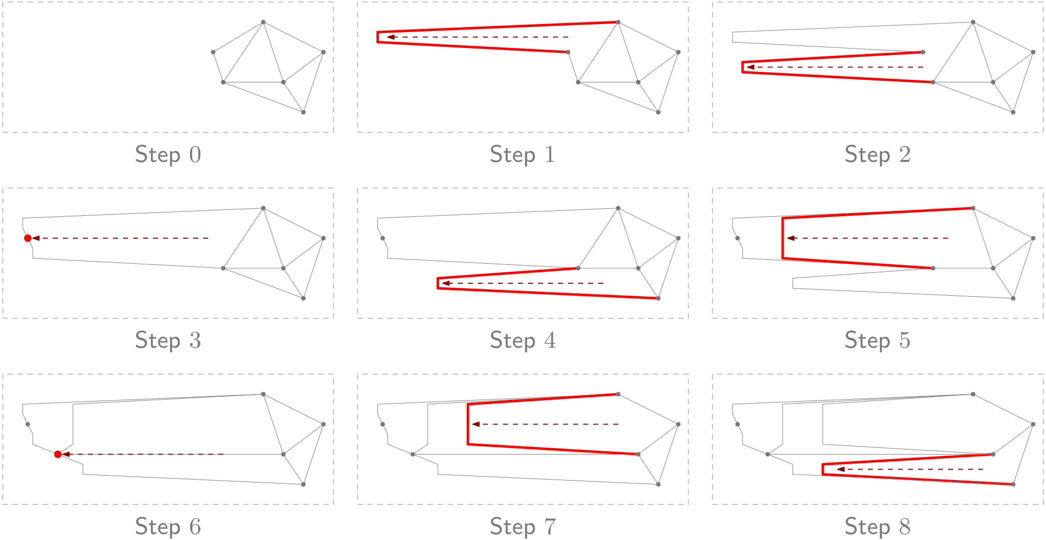
A visibility representation that can be computed from P.



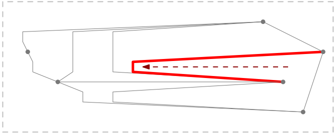
An orthogonal box drawing, and corresponding admitted drawing P', which can both be computed from P.

How do we actually perform a morph?

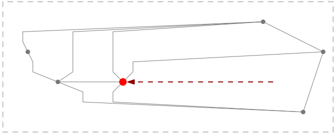
Morphing from a straight-line to an admitted drawing: Method



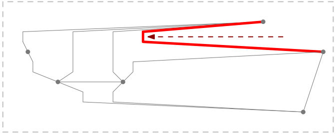
Morphing from a straight-line to an admitted drawing: Method



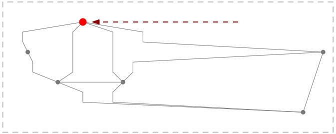
Step 9



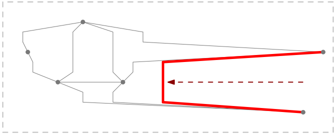
Step 10



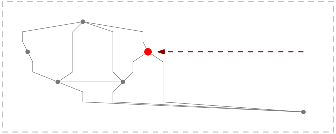
Step 11



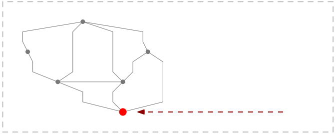
Step 12



Step 13



Step 14



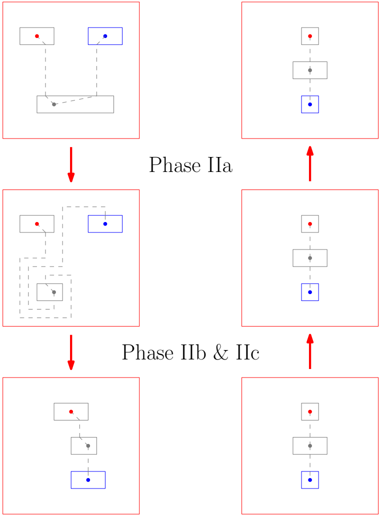
Step 15

Phase II

High-level: Reduce to parallel box drawing morphs (only lengths differ). Only need to morph (not a different drawing type).

Phase II Overview

- **Input:** Orthogonal box drawing pair
- **Output:** Parallel orthogonal box drawing pair (for each edge: same port locations, same sequence of turns)
- Substeps:
  - Phase IIa: Adjust port locations
  - Phase IIb: Global analysis $\mapsto$ instructions
  - Phase IIc: Instructions $\mapsto$ local changes

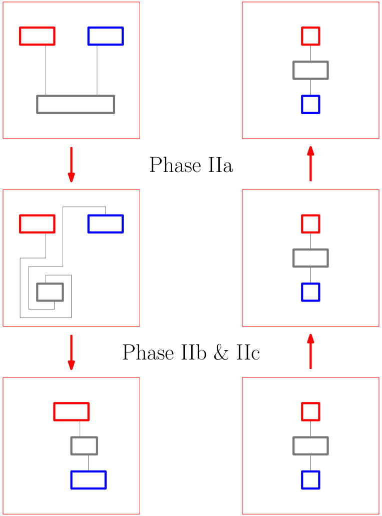


Phase II

High-level: Reduce to parallel box drawing morphs (only lengths differ). Only need to morph (not a different drawing type).

Phase II Overview

- **Input:** Orthogonal box drawing pair
- **Output:** Parallel orthogonal box drawing pair (for each edge: same port locations, same sequence of turns)
- Substeps:
  - Phase IIa: Adjust port locations
  - Phase IIb: Global analysis $\mapsto$ instructions
  - Phase IIc: Instructions $\mapsto$ local changes

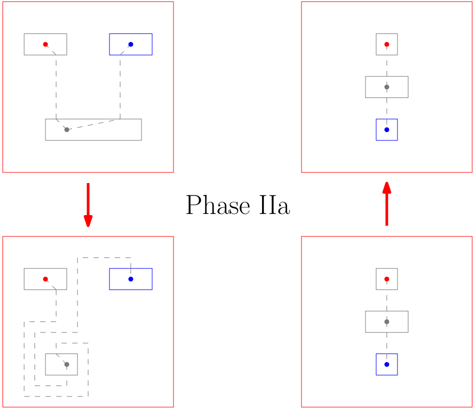


Phase IIa

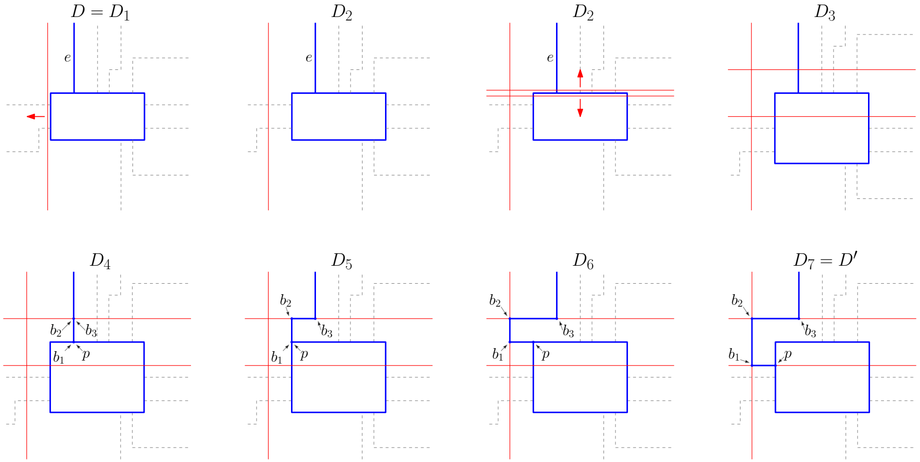
High-level: Move ports. Add bends to do so.

Phase IIa Overview

- **Input:** Orthogonal box drawing pair
- **Output:** Port-aligned orthogonal box drawing pair (same relative port locations)

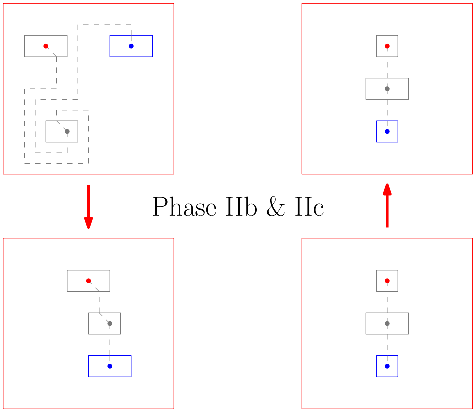


Moving Ports around Corners



Phase IIb

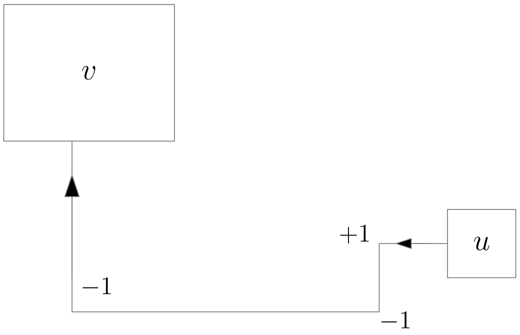
High-level: Do some (global) analysis on the edges.



Phase IIb Overview

- **Input:** Port-aligned orthogonal box drawing pair
- **Output:** "Instructions"

Spirality

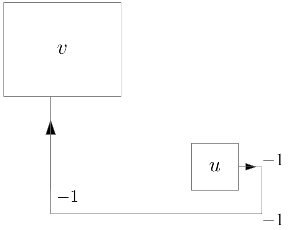


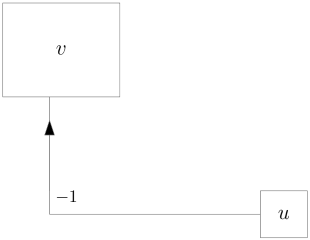
Spirality of the edge uv (oriented u to v): -1.

Difference in Spirality (1)

Difference in spirality of the edge uv (oriented u to v): -2.

Difference in Spirality (1)

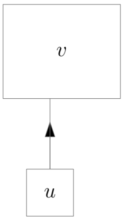


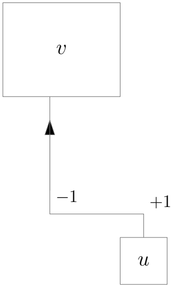


Difference in spirality of the edge uv (oriented u to v): -2.

Goal: Reduce this to zero.

Difference in Spirality (2)

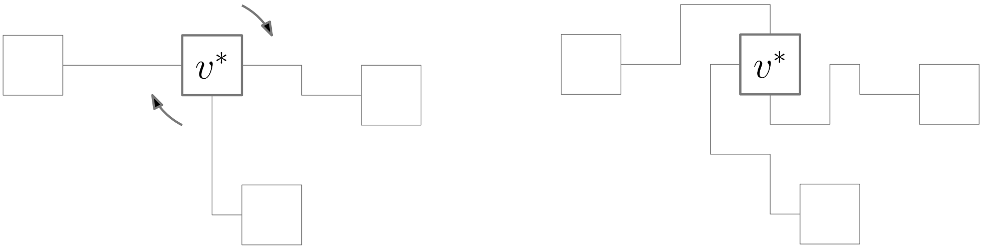




Difference in spirality of the edge uv (oriented u to v): 0.

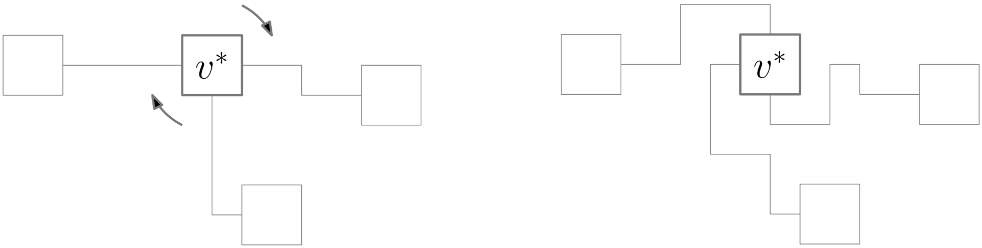
Goal: Reduce this to zero.

Twists (High-Level)



Spirality changes! Net turns are added.
Don't know how to compute these drawings yet (happens in Phase IIc).

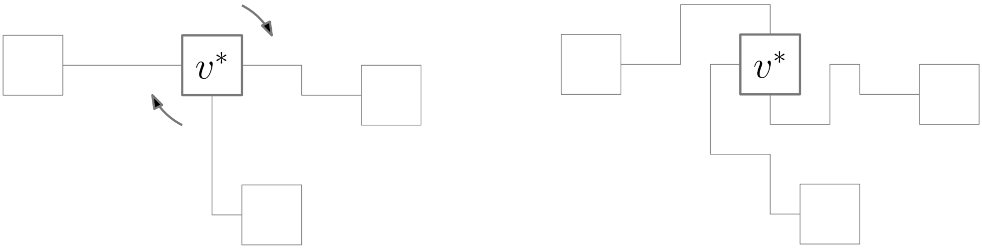
Twists (High-Level)



Spirality changes! Net turns are added.
Don't know how to compute these drawings yet (happens in Phase IIc).

Similar to a result by Biedl et al.: Exists some number/direction of twists for each vertex so that difference in spirality becomes zero everywehere. This number is $O(n)$ for each vertex.

Twists (High-Level)



Spirality changes! Net turns are added.
Don't know how to compute these drawings yet (happens in Phase IIc).

Similar to a result by Biedl et al.: Exists some number/direction of twists for each vertex so that difference in spirality becomes zero everywehere. This number is $O(n)$ for each vertex.

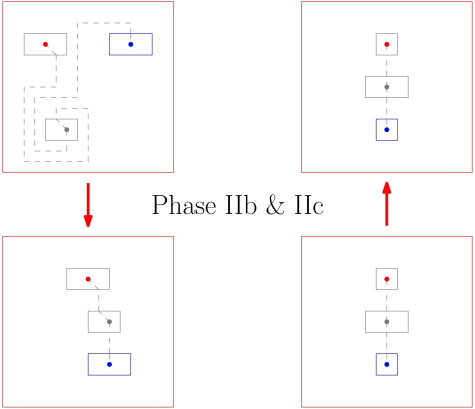
Key difference/contribution: We use simultaneous twists, so only $O(n)$ operations needed.

Phase IIc

High-level: Use analysis to get rid of bends.

Phase IIc Overview

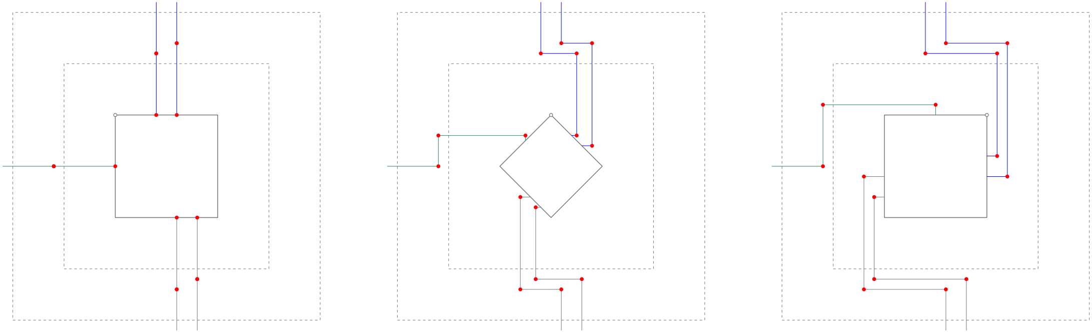
- **Input:** Port-aligned orthogonal box drawing pair, simultaneous twist instructions
- **Output:** Parallel orthogonal box drawings
- Two components:
  - Perform twists
  - Obtain canonical drawings



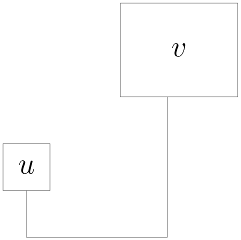
Twists

Two steps:

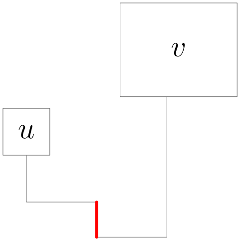
- "Prepare" drawing (make boxes square, well-spaced out)
- Twist everything simultaneously



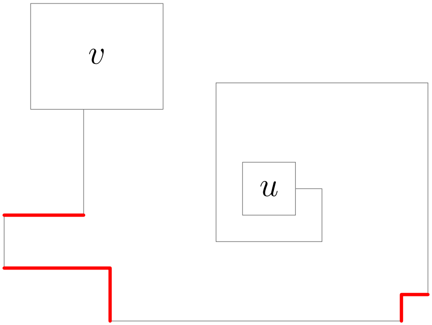
Simplification/Canonical form: Zig-Zags



No zig-zags.



One (vertical) zig-zag.

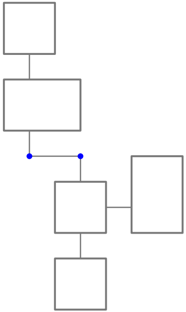


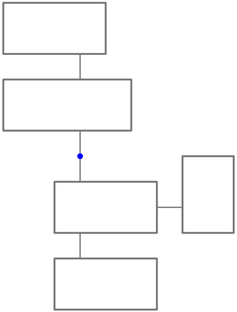
Five zig-zags, three horizontal and two vertical.

We want to remove zig-zags.

Simplification/Canonical form: Removing a Single Zig-Zag (1)

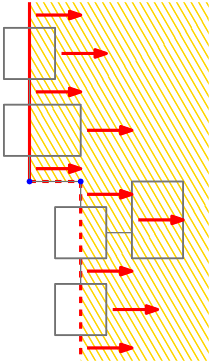
Method by Biedl et al.:

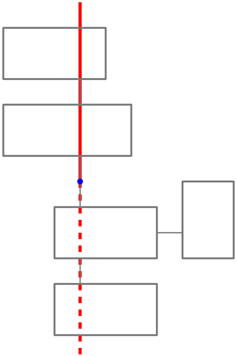




This is a unidirectional morph.

Simplification/Canonical form: Removing a Single Zig-Zag (2)





Push each thing over if it lies to the right of the divider.

Simplification/Canonical form: Removing a Single Zig-Zag (2)

Push each thing over if it lies to the right of the divider.

Problem: Requires a morph for each zig-zag (want $O(1)$ morphs for all zig-zags).

Simplification/Canonical form: Removing a Single Zig-Zag (2)

Push each thing over if it lies to the right of the divider.

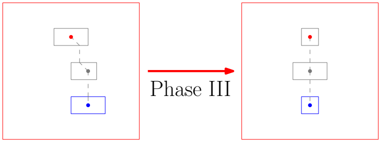
Problem: Requires a morph for each zig-zag (want $O(1)$ morphs for all zig-zags).
Solution/new contribution: $O(1)$ morphs suffice, even on a grid (skipping details).

Phase III

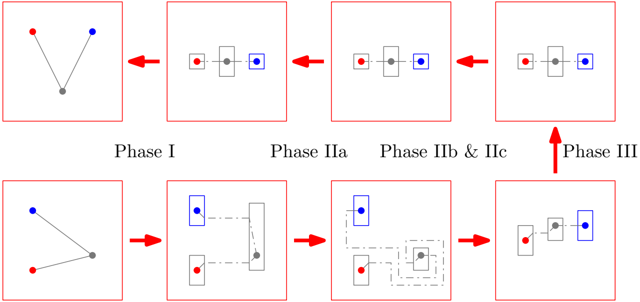
High-level: Use black-box result to morph parallel orthogonal box drawings (i.e., adjust lengths).

Phase III Overview

- **Input:** Parallel orthogonal box drawing pair.
- **Output:** Linear morph sequence.
- **Methodology:** Appeal to black-box result by Biedl et al.. It requires connectivity.
  - Essentially, add edges to both drawings (and simplify again) until every face is a rectangle.



	Graph/drawing class	Num. linear morphs	Grid size side-length	Bends per edge	Time complexity
Main result	Connected	$O(n)$	$O(n)$	$O(1)$	$O(n^2)$
Main method	Connected Ortho-Box	$O(n)$	$O(n)$	$O(1)$	$O(n^2)$



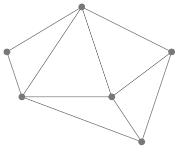
Open problems:

- Seemingly approachable: Unidirectional morphs only, max 4 bends, max 2 bends, disconnected graphs.
- Big: No bends.

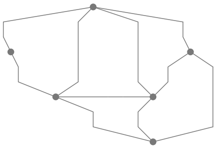
Fin.

Morphing from a straight-line to an admitted drawing: Brainstorming (1)

Have: a planar straight-line drawing P, an orthogonal box drawing D with an admitted drawing P'.
Want: Morph from P to P'. Bends need to be added.

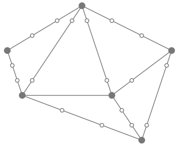


P



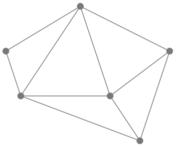
P'

- Idea 1: Use same y-coordinate
- Problem: Not integer coordinates

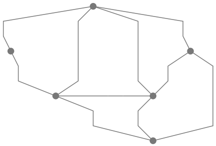


Morphing from a straight-line to an admitted drawing: Brainstorming (2)

Have: a planar straight-line drawing P, an orthogonal box drawing D with an admitted drawing P'.
Want: Morph from P to P'. Bends need to be added.



P



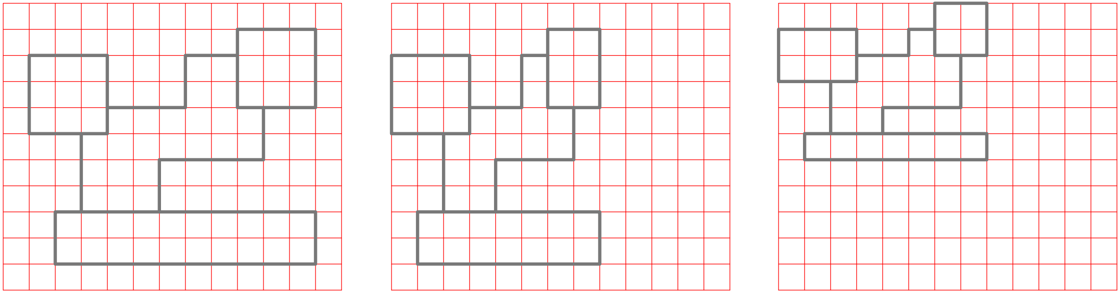
P'

- ~~Idea 1: Use same y-coordinate~~
- Idea 2: Make them coincident with the vertex
- Possible problem: Not a unidirectional morph (complicated movement).
- Alleviation: Perform the morph on one vertex/edge at a time.

Compressions

Want to be able to bring a drawing to an $O(n) \times O(n)$ grid from an arbitrarily sized grid (where the constant is independent of the initial grid size).

Idea: Sort x-coordinates.



This is a unidirectional morph.

Simplification—Removing all Horizontal Zig-Zags (High-level)

Each problem has a different solution:

- Requires a morph for each zig-zag (want $O(1)$ morphs for all zig-zags).
  - Van Goethem et al.: A single morph suffices for many (disjoint) horizontal zig-zags.

Simplification—Removing all Horizontal Zig-Zags (High-level)

Each problem has a different solution:

- Requires a morph for each zig-zag (want $O(1)$ morphs for all zig-zags).
  - Van Goethem et al.: A single morph suffices for many (disjoint) horizontal zig-zags. Two issues with their solution:
    - Uses a large grid.
    - Slow time complexity.

Simplification—Removing all Horizontal Zig-Zags (High-level)

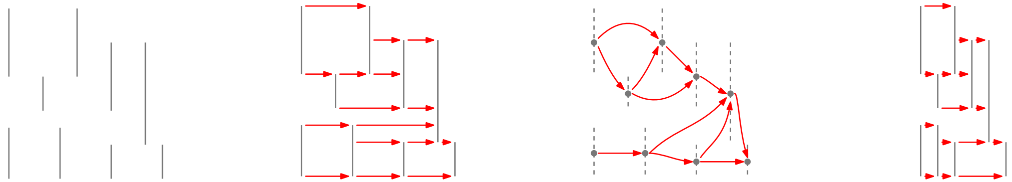
Each problem has a different solution:

- Requires a morph for each zig-zag (want $O(1)$ morphs for all zig-zags).
  - Van Goethem et al.: A single morph suffices for many (disjoint) horizontal zig-zags. Two issues with their solution:
    - Uses a large grid.
    - Slow time complexity.
- Requires $O(n)$ time for each zig-zag (want $O(n)$ time for all zig-zags).
  - Use circuit layout compaction!

Simplification—Circuit Compaction

Goal: Compress vertical line segments.

Solution by Doenhardt and Lengauer:

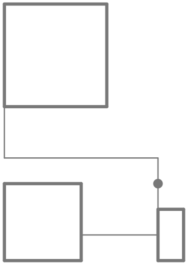


(1) Input (2) Trapezoidal Map (3) Trapezoidal Graph (4) Result from topological sort

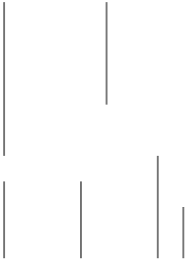
Important note:
Last step of Doenhardt and Lengauer's algorithm only needs y-coordinates and trapezoidal graph.

Simplification—Circuit Compaction for Box Drawings

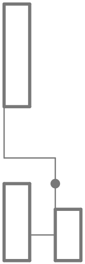
Goal: Compress a box drawing (again).



A box drawing C



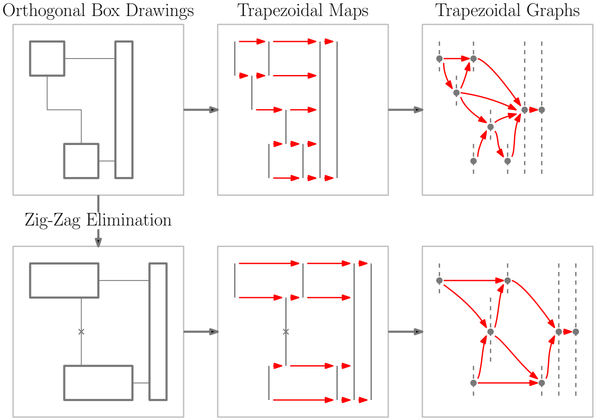
A set of maximal vertical line segments $L(C)$ covering C



The compressed drawing C'

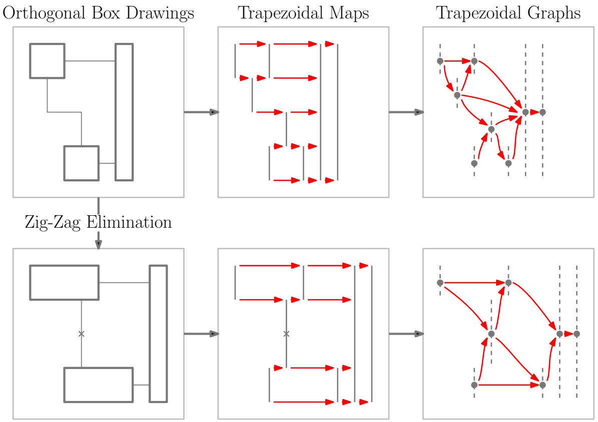
Side note: Doing this in $O(n)$ time requires connectivity (via an algorithm by Chazelle for trapezoidal maps of simple polygons).

Simplification—Zig-Zag Elimination and Circuit Compaction



Takeaway: The changes to the trapezoidal graph are local to the zig-zag being eliminated.

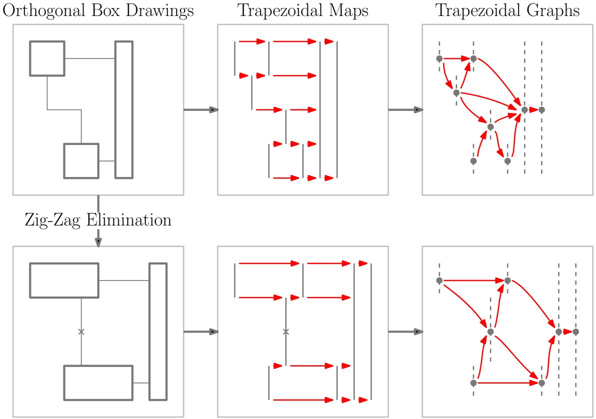
Simplification—Zig-Zag Elimination and Circuit Compaction



Takeaway: The changes to the trapezoidal graph are local to the zig-zag being eliminated.

Recall: Last step of Doenhardt and Lengauer's algorithm only needs y-coordinates and trapezoidal graph.

Simplification—Zig-Zag Elimination and Circuit Compaction

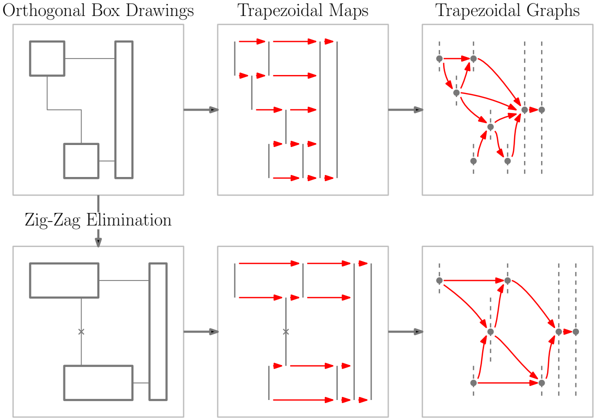


Takeaway: The changes to the trapezoidal graph are local to the zig-zag being eliminated.

Recall: Last step of Doenhardt and Lengauer's algorithm only needs y-coordinates and trapezoidal graph.

Idea: Compute only the trapezoidal graph after a sequence of zig-zag eliminations.

Simplification—Zig-Zag Elimination and Circuit Compaction



Takeaway: The changes to the trapezoidal graph are local to the zig-zag being eliminated.

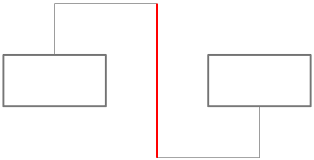
Recall: Last step of Doenhardt and Lengauer's algorithm only needs y-coordinates and trapezoidal graph.

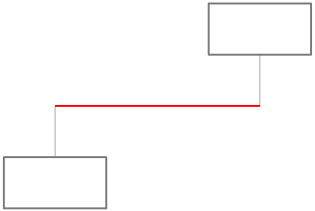
Idea: Compute only the trapezoidal graph after a sequence of zig-zag eliminations.

Final result: Can remove all horizontal zig-zags in one linear morph, in $O(n)$ time.

Simplification—Eliminating All Zig-Zags

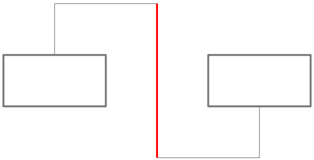
Eliminating all horizontal zig-zags $\neq$ eliminating all zig-zags:

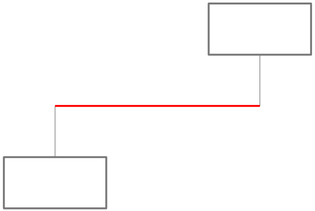




Simplification—Eliminating All Zig-Zags

Eliminating all horizontal zig-zags $\neq$ eliminating all zig-zags:





Eliminating all horizontal (and then vertical) zig-zags does reduce the number of bends per edge (unless there are no zig-zags).

Simplification—Eliminating All Zig-Zags

Eliminating all horizontal zig-zags $\neq$ eliminating all zig-zags:

Eliminating all horizontal (and then vertical) zig-zags does reduce the number of bends per edge (unless there are no zig-zags).

Idea: Since $O(1)$ bends per edge is maintained, only need to do $O(1)$ simultaneous eliminations to eliminate all zig-zags.

Phase II High-Level