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

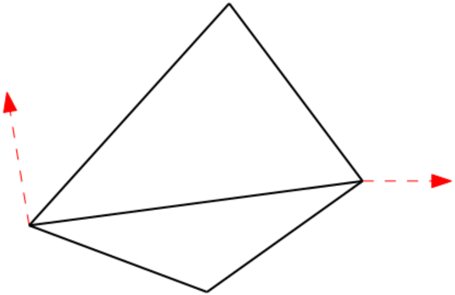
Graph Drawing 2024

Morphing Planar Graph Drawings via Orthogonal Box Drawings

Joint work with Therese Biedl and Anna Lubiw

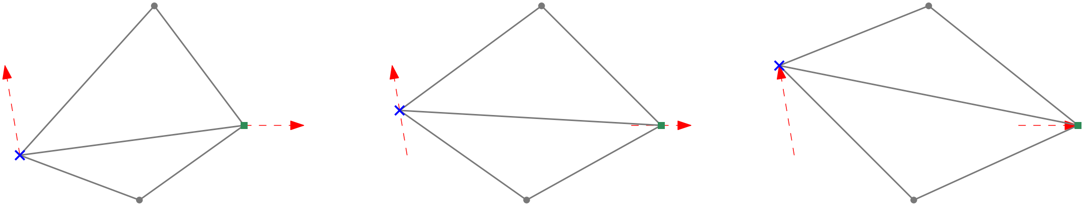
Graph Morphing

Morph: Continuously transform between drawings (with time $t \in [0, 1]$)



Graph Morphing

Morph: Continuously transform between drawings (with time $t \in [0, 1]$)



$t = 0$

P

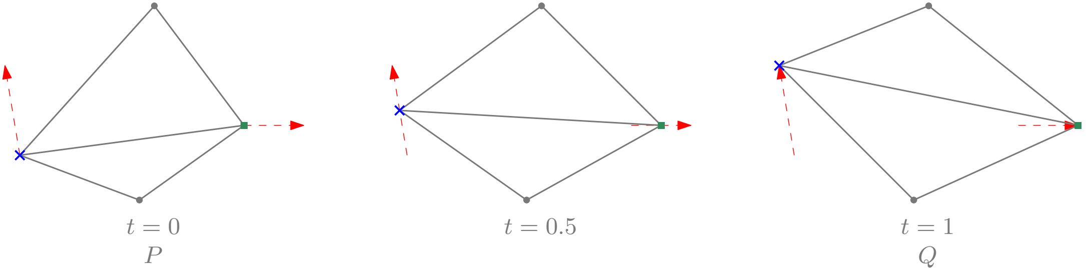
$t = 0.5$

$t = 1$

Q

Graph Morphing

Morph: Continuously transform between drawings (with time $t \in [0, 1]$)



Linear morph: Linearly interpolate vertex (and other) locations

Goals of morphing

Goal I: Make “nice” morphs.

- Simple paths of movement.
- Elementary steps.
- Few steps.

Goals of morphing

Goal I: Make “nice” morphs.

- Simple paths of movement.
- Elementary steps.
- Few steps.

Goal II: Preserve drawing properties!
Planarity, few bend/straight line edges, orthogonality, drawing on a small grid, etc. Two types:

- Between elementary steps.
- During elementary steps/at all times.

Goals of morphing

Goal I: Make "nice" morphs.

- Simple paths of movement.
- Elementary steps.
- Few steps.

Goal II: Preserve drawing properties!
Planarity, few bend/straight line edges, orthogonality, drawing on a small grid, etc. Two types:

- Between elementary steps.
- During elementary steps/at all times.

Goal III: Algorithmic properties

- Time complexity
- Computational Model

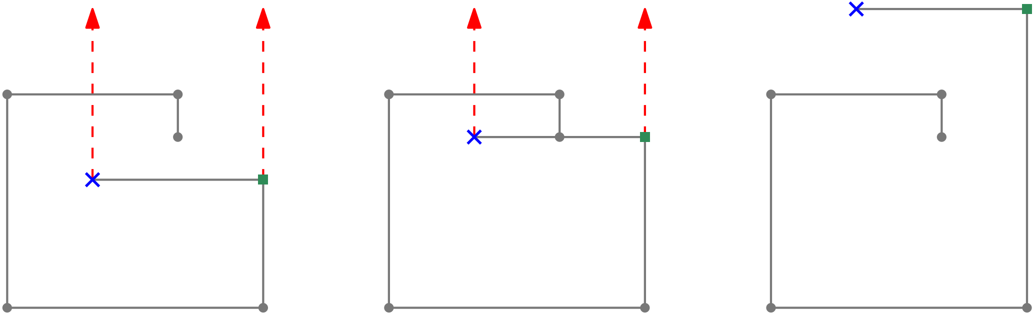
Planarity-Preserving Morphs (preserved at all times)

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

Planarity-Preserving Morphs (preserved at all times)

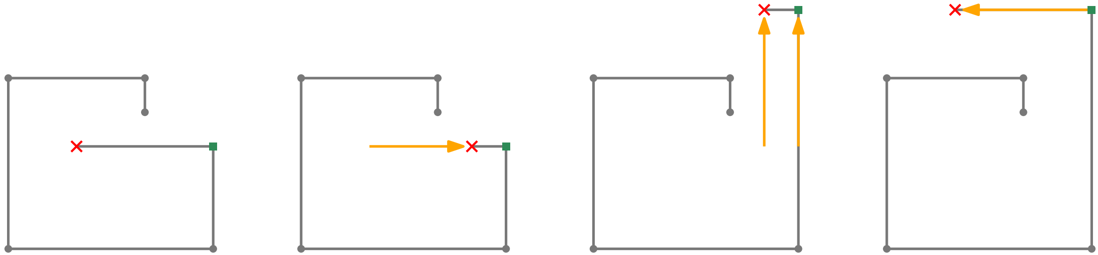
Planarity-Preserving Morph: At all times t, the "interpolated" drawing is also planar.

Non-planarity-preserving morph:



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

Linear Morphs Sequences



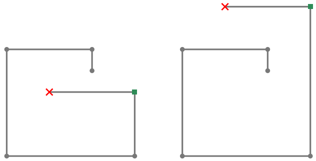
$P = D_1$ D_2 D_3 $D_4 = Q$

These are **explicit intermediate drawings**.
Entire morph represented by a finite sequence D_1, D_2, D_3, D_4.

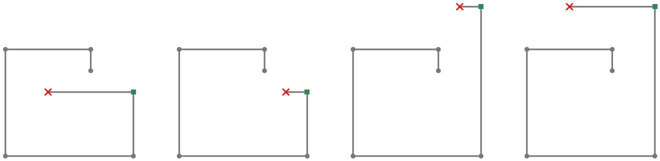
Can preserve extra properties on these drawings ("between steps").

The Linear Morph Problem

Input:
'Compatible' pair of drawings (labelled)

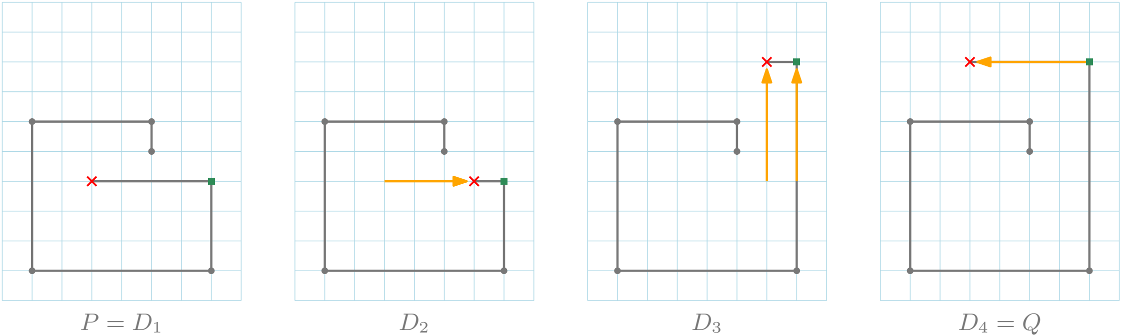


Output:
Planarity-preserving linear morph sequence (list of drawings)



Objectives: Numerous!

Linear Morph Sequences on a Grid



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

Open Problem in Morphing

Open:
Input: Straight-line drawings each on an $O(n) \times O(n)$ grid (same embedding of same graph).
Output: Linear morph sequence with properties:

- Explicit drawings: Polynomial-sized grid.
- Implicit+explicit drawings: Planar, straight-line edges
- Linear morph sequence length: Polynomial.

Open Problem in Morphing

Open:
Input: Straight-line drawings each on an $O(n) \times O(n)$ grid (same embedding of same graph).
Output: Linear morph sequence with properties:

- Explicit drawings: Polynomial-sized grid.
- Implicit+explicit drawings: Planar, straight-line edges
- Linear morph sequence length: Polynomial.

Conjecture: Even stronger properties can be obtained.
Input: Straight-line drawings each on an $O(n) \times O(n)$ grid (same embedding of same graph).
Output: Linear morph sequence with properties:

- Explicit drawings: $O(n) \times O(n)$ grid.
- Implicit+explicit drawings: Planar, straight-line edges
- Linear morph sequence length: $O(n)$

Open Problem in Morphing

Open:
Input: Straight-line drawings each on an $O(n) \times O(n)$ grid (same embedding of same graph).
Output: Linear morph sequence with properties:

- Explicit drawings: Polynomial-sized grid.
- Implicit+explicit drawings: Planar, straight-line edges
- Linear morph sequence length: Polynomial.

Conjecture: Even stronger properties can be obtained.
Input: Straight-line drawings each on an $O(n) \times O(n)$ grid (same embedding of same graph).
Output: Linear morph sequence with properties:

- Explicit drawings: $O(n) \times O(n)$ grid.
- Implicit+explicit drawings: Planar, straight-line edges
- Linear morph sequence length: $O(n)$

Various weakenings are known. We present a new one.

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.

Previous Results (Abridged)

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 (Abridged)

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 (Abridged)

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 (Abridged)

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 (Abridged)

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

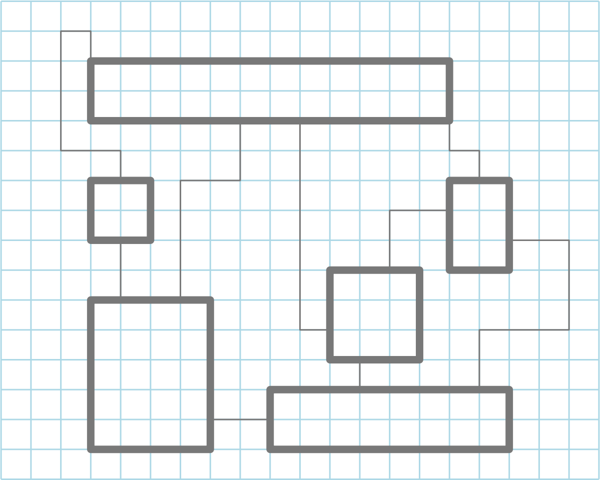
	Graph/Drawing Class	Num linear morphs	Grid-size side-length	Bends per edge	Comput. Model	Time Complexity
Alamdari et al. (2017)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^3)$
Klemz (2021)	Straight-line	$O(n)$	Expo.	0	Powerful	$O(n^2 \log n)$
Klemz (2021)	2-connected	$O(n)$	Expo.	0	Powerful	$O(n^2)$
Lubiw & 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	$O(n^2)$
Open	Many	Poly	Poly	0	Any	Any
Lower Bounds	Planar	$O(n)$	$O(n)$	0	Word RAM	$O(n^2)$

Grid size assumes input fits on the same grid.

Above table is not comprehensive.

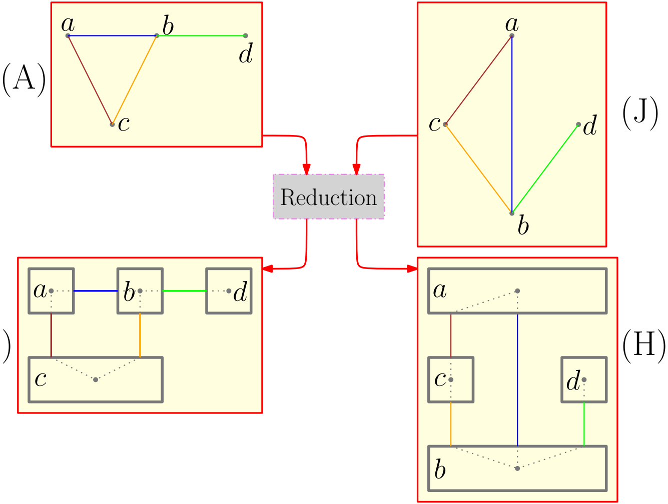
High-Level Overview

First: Reduce problem to morphing **orthogonal box drawings**.



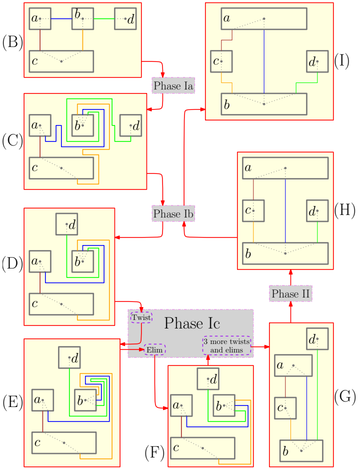
High-Level Overview

First: Reduce problem to morphing "orthogonal box drawings".

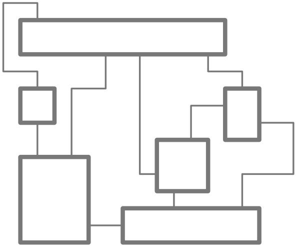


High-Level Overview

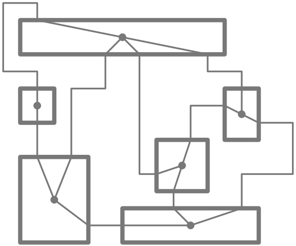
Second: Morph orthogonal box drawings.



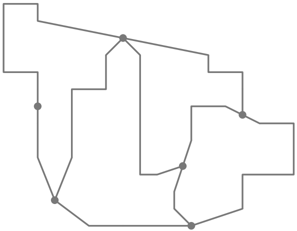
Reduction: Admitted Drawings (1)



Orthogonal box drawing

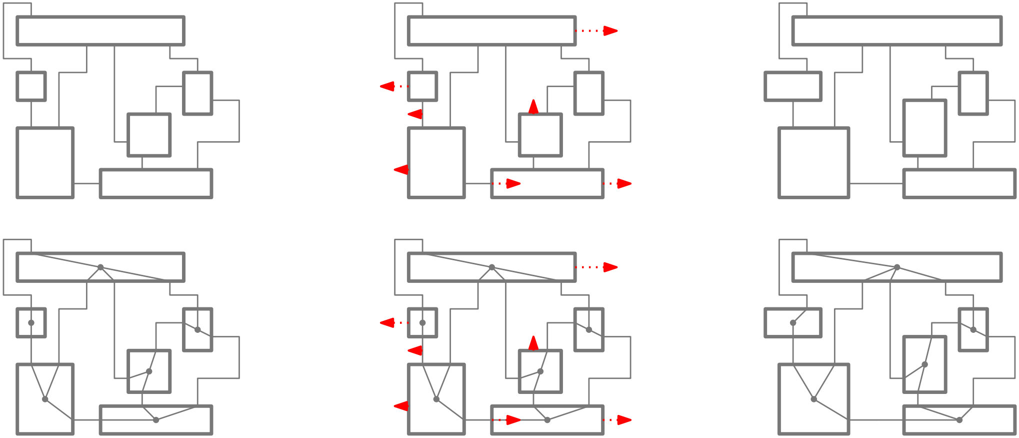


Both



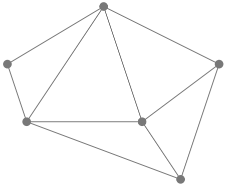
Admitted poly-line drawing

Reduction: Admitted Drawings (2)

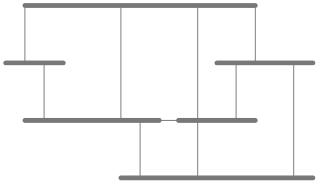


Morph of orthogonal box drawings $\implies$ morph of admitted drawings

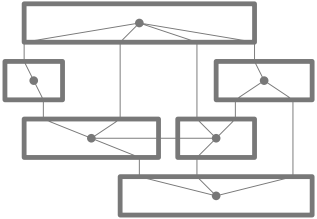
Computing Box Drawings: Visibility Representations as an Intermediary



A planar straight-line drawing P.

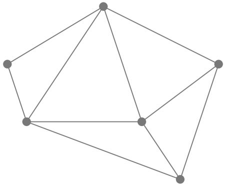


A visibility representation that can be computed from P.

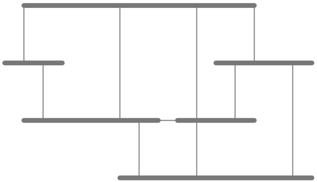


An orthogonal box drawing, and corresponding admitted drawing P', which can both be computed from P.

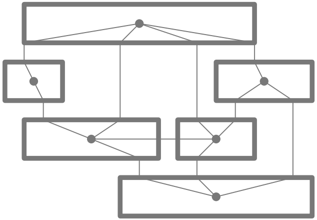
Computing Box Drawings: Visibility Representations as an Intermediary



A planar straight-line drawing P.



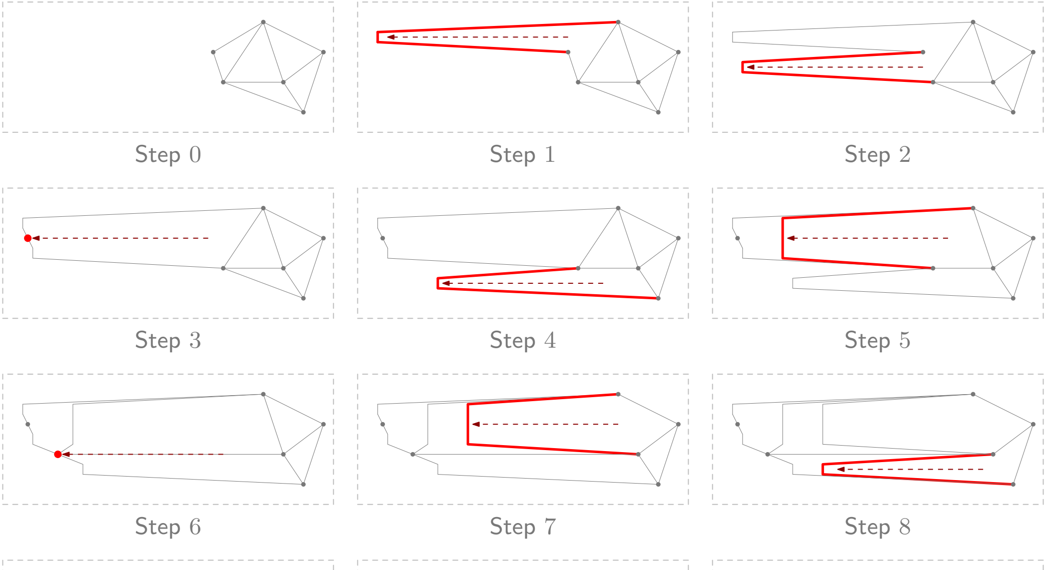
A visibility representation that can be computed from P.



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

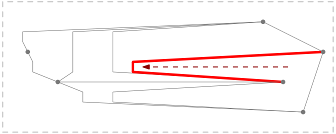


Step 0 Step 1 Step 2

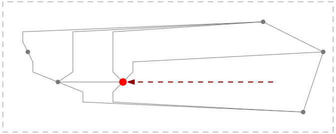
Step 3 Step 4 Step 5

Step 6 Step 7 Step 8

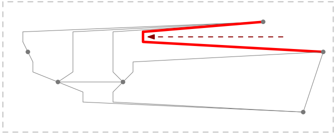
Morphing from a straight-line to an admitted drawing: Method



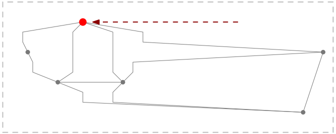
Step 9



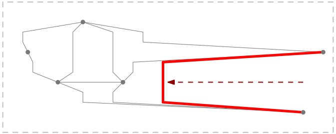
Step 10



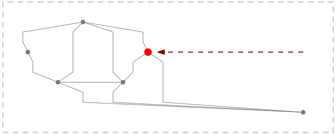
Step 11



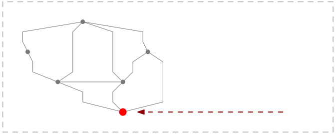
Step 12



Step 13



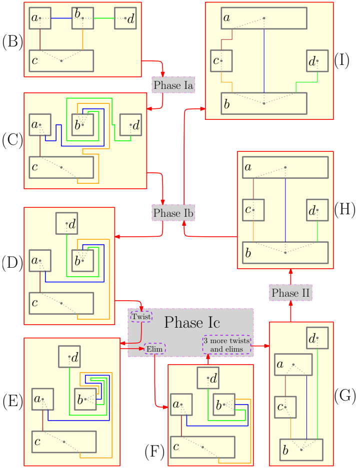
Step 14



Step 15

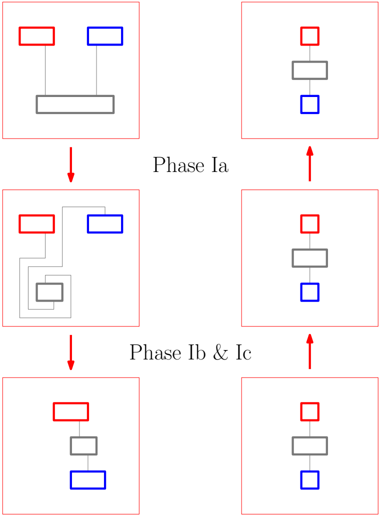
Recall: High-Level Overview

Now have orthogonal box drawings, want to morph them.



Phase I

Goal: Reduce to parallel box drawings.

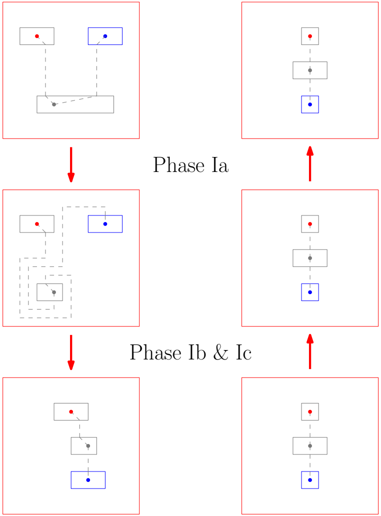


Phase I

Goal: Reduce to parallel box drawings.

Phase I Overview

- **Input:** Orthogonal box drawing pair
- **Output:** Parallel orthogonal box drawing pair (for each edge: same port locations, same sequence of turns)
- Substeps:
  - Phase Ia: Adjust port locations
  - Phase Ib: Initial zig-zag elimination
  - Phase Ic: Twists (plus more interspersed compaction/zig-zag elim)

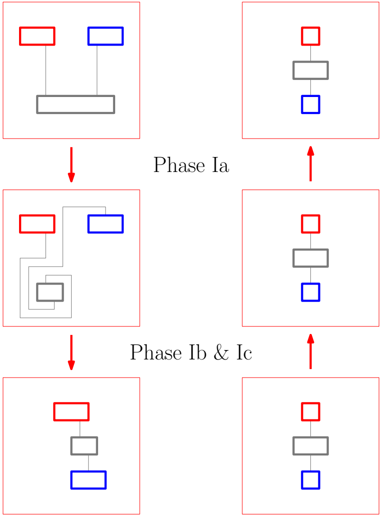


Phase I

Goal: Reduce to parallel box drawings.

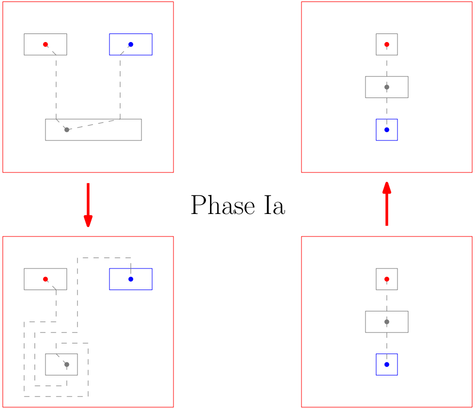
Phase I Overview

- **Input:** Orthogonal box drawing pair
- **Output:** Parallel orthogonal box drawing pair (for each edge: same port locations, same sequence of turns)
- Substeps:
  - Phase Ia: Adjust port locations
  - Phase Ib: Initial zig-zag elimination
  - Phase Ic: Twists (plus more interspersed compaction/zig-zag elim)



Phase Ia

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

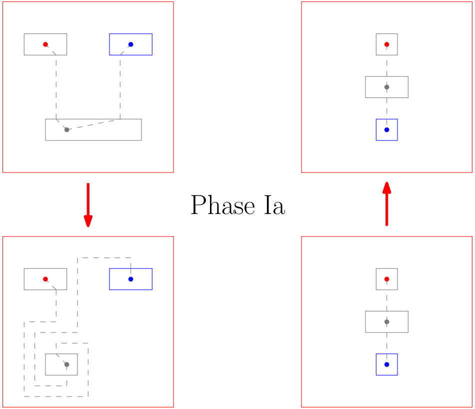


Phase Ia

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

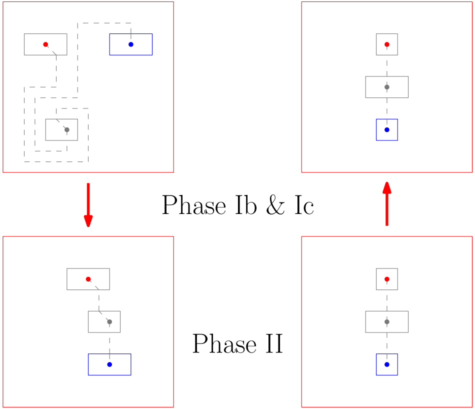
Phase Ia Overview

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



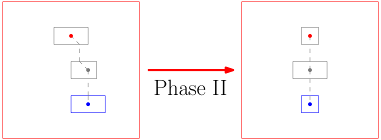
Phase Ib & 1c

Get rid of all (extra) bends.

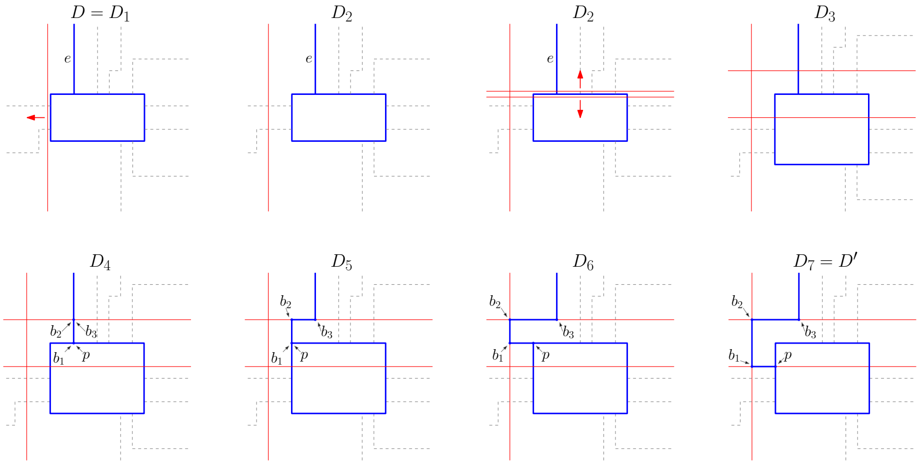


Phase II

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



Moving Ports around Corners

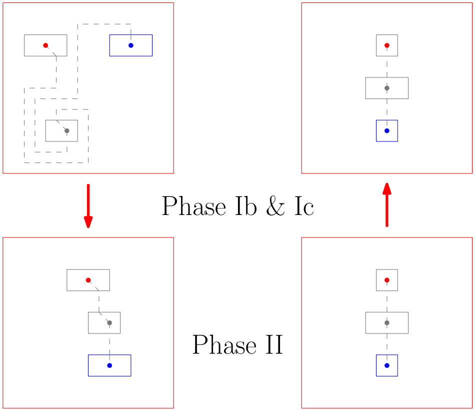


Phase Ib & 1c

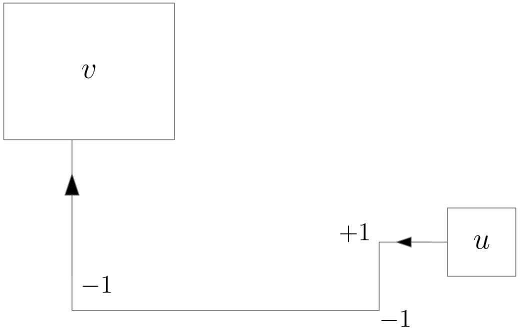
Get rid of all (extra) bends.

Phase Ib Overview

- **Input:** Port-aligned orthogonal box drawing pair
- **Output:** Parallel orthogonal box drawing pair

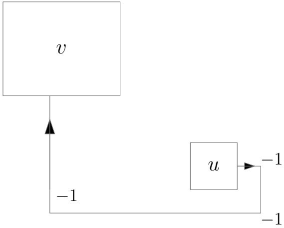


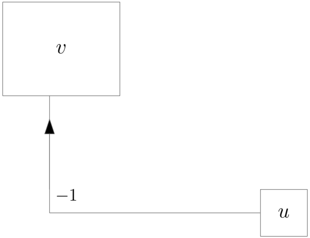
Spirality



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

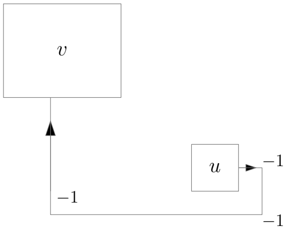
Difference in Spirality (1)

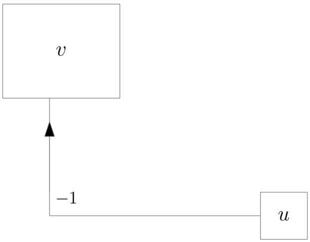




Difference in spirality of the edge uv (oriented u to v): -2.

Difference in Spirality (1)

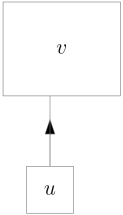


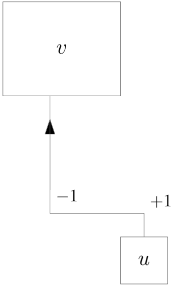


Difference in spirality of the edge uv (oriented u to v): -2.

Goal: Reduce this to zero.

Difference in Spirality (2)

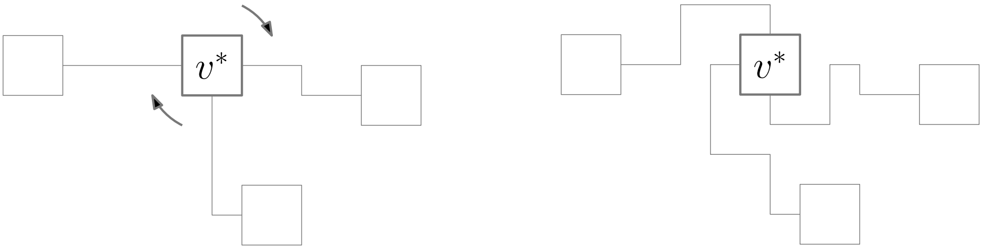




Difference in spirality of the edge uv (oriented u to v): 0.

Goal: Reduce this to zero.

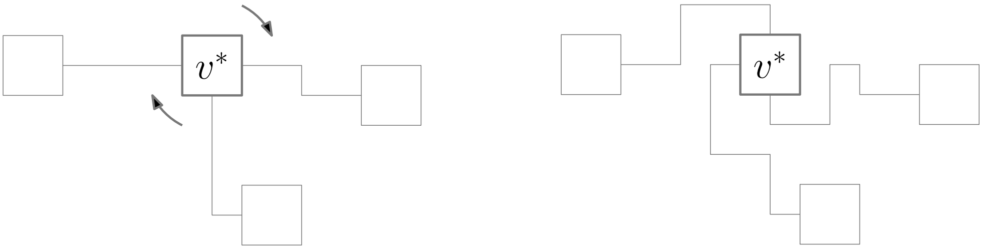
Twists (High-Level)



Spirality changes! Net turns are added.

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.

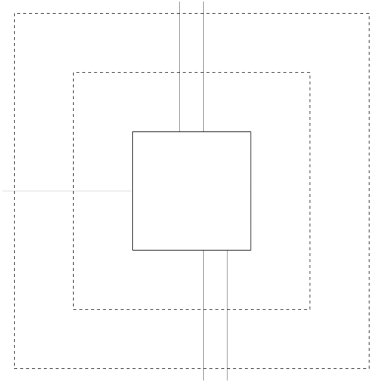
Similar to a result by Biedl et al.: Exists some number/direction of twists for each vertex so that difference in spirality becomes zero everywehere. This number is $O(n)$ for each vertex.

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

Twists (Implementation)

Two steps:

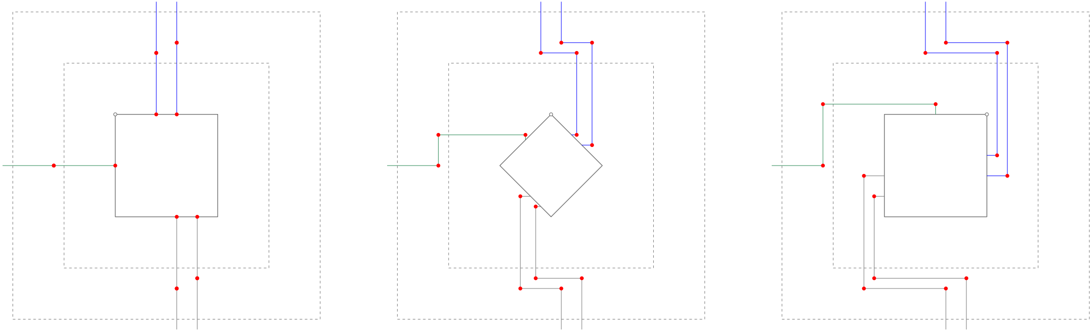
- "Prepare" drawing (make boxes square, well-spaced out)
- Twist everything simultaneously



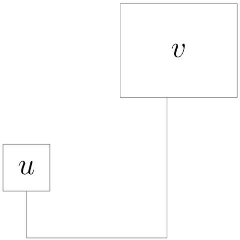
Twists (Implementation)

Two steps:

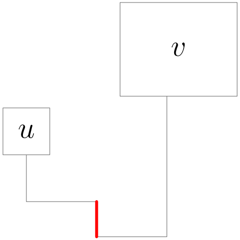
- "Prepare" drawing (make boxes square, well-spaced out)
- Twist everything simultaneously



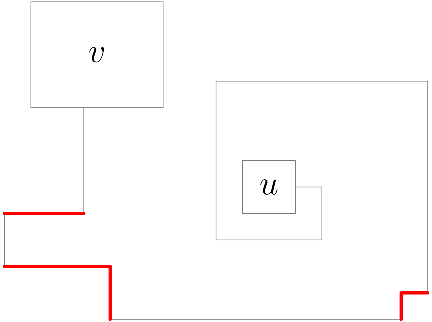
Simplification/Canonical form: Zig-Zags



No zig-zags.



One (vertical) zig-zag.

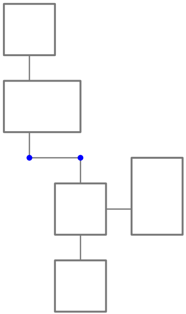


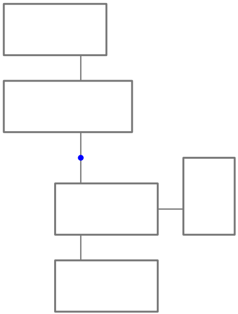
Five zig-zags, three horizontal and two vertical.

We want to remove zig-zags.

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

Method by Biedl et al.:



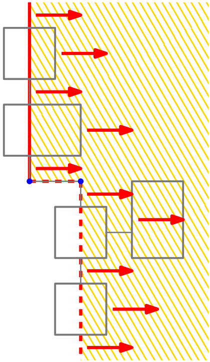


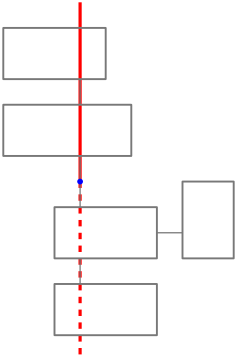
This is a unidirectional morph.

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.

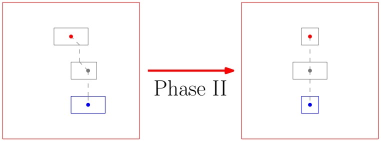
Problem: Requires a morph for each zig-zag (want $O(1)$ morphs for all zig-zags).
Solution/new contribution: $O(1)$ morphs suffice, even on a grid (skipping details).

Phase II

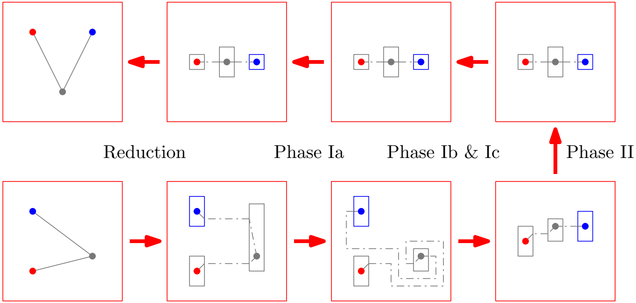
High-level: Use black-box result to morph parallel orthogonal box drawings (i.e., adjust lengths).

Phase II 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.



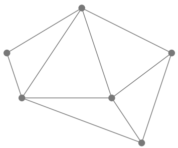
	Graph/Drawing Class	Num linear morphs	Grid-size side-length	Bends per edge	Comput. Model	Time Complexity
Main result	Connected	$O(n)$	$O(n)$	$O(1)$	Word RAM	$O(n^2)$
This work (main method)	Connected Ortho-Box	$O(n)$	$O(n)$	$O(1)$	Word RAM	$O(n^2)$
Open	Many	Poly	Poly	0	Any	Any
Lower Bounds	Planar	$O(n)$	$O(n)$	0	Word RAM	$O(n^2)$



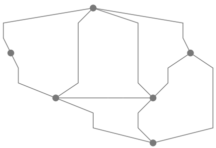
Fin

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.

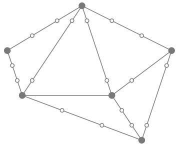


P



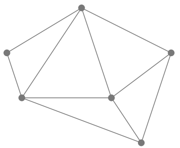
P'

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

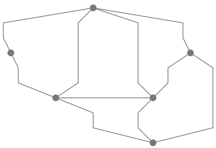


Morphing from a straight-line to an admitted drawing: Brainstorming (2)

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



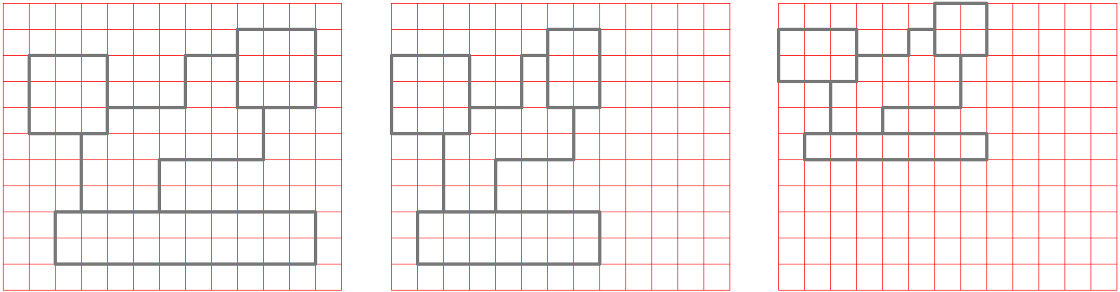
P'

- ~~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)

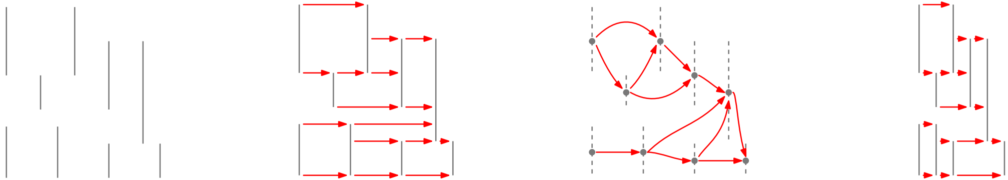
Each problem has a different solution:

- 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:

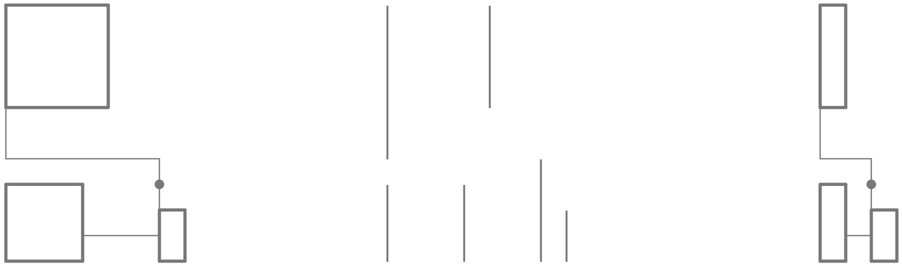


(1) Input (2) Trapezoidal Map (3) Trapezoidal Graph (4) Result from topological sort

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).



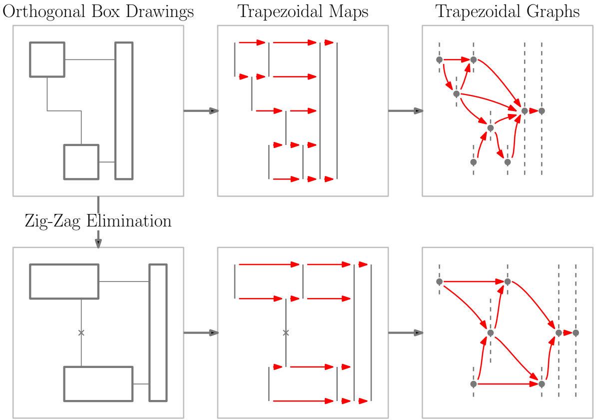
A box drawing C

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

The compressed drawing C'

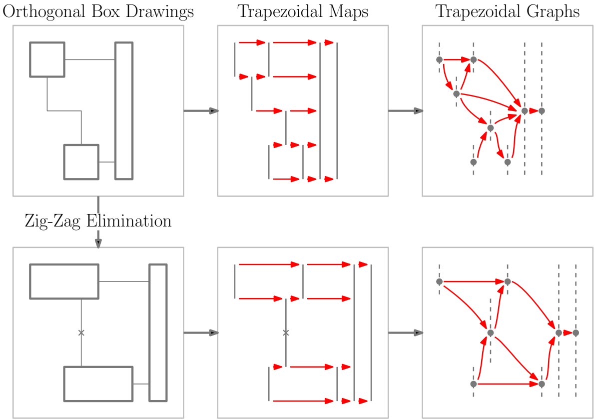
Side note: Doing this in $O(n)$ time requires connectivity (via an algorithm by Chazelle for trapezoidal maps of simple polygons).

Simplification—Zig-Zag Elimination and Circuit Compaction



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

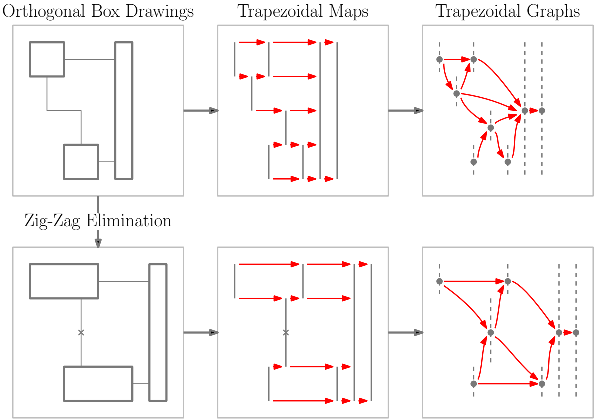
Simplification—Zig-Zag Elimination and Circuit Compaction



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

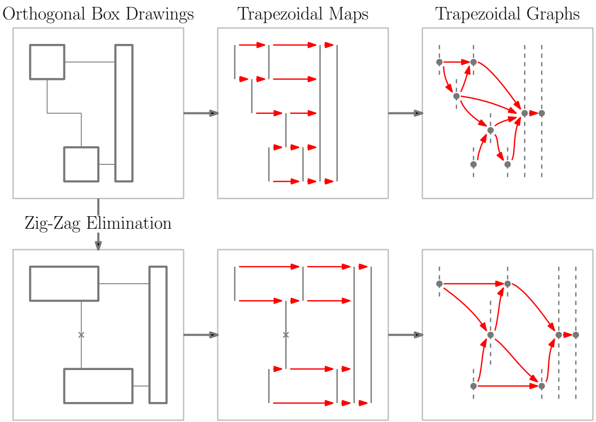


Takeaway: The changes to the trapezoidal graph are local to the zig-zag being eliminated.

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.

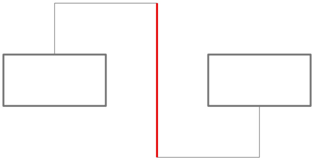
Recall: Last step of Doenhardt and Lengauer's algorithm only needs y-coordinates and trapezoidal graph.

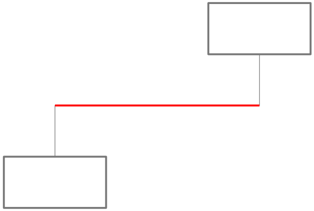
Idea: Compute only the trapezoidal graph after a sequence of zig-zag eliminations.

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 I High-Level