Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

Layered Drawing of Undirected Graphs with Generalized Port Constraints

Julian Walter, **Johannes Zink**,
Joachim Baumeister, Alexander Wolff

Motivation

- Imagine you own a machine manufacturing company.

Motivation

- Imagine you own a machine manufacturing company.
- The company produces various models with different equipment and individual custom-made parts.

Motivation

- Imagine you own a machine manufacturing company.
- The company produces various models with different equipment and individual custom-made parts.
- In the case of a malfunction, a technician needs a drawn cable plan to understand the particular interdependencies.

Motivation

- Imagine you own a machine manufacturing company.
- The company produces various models with different equipment and individual custom-made parts.
- In the case of a malfunction, a technician needs a drawn cable plan to understand the particular interdependencies.
- Hand-drawn plans for all variants are unreasonably expensive.

Motivation

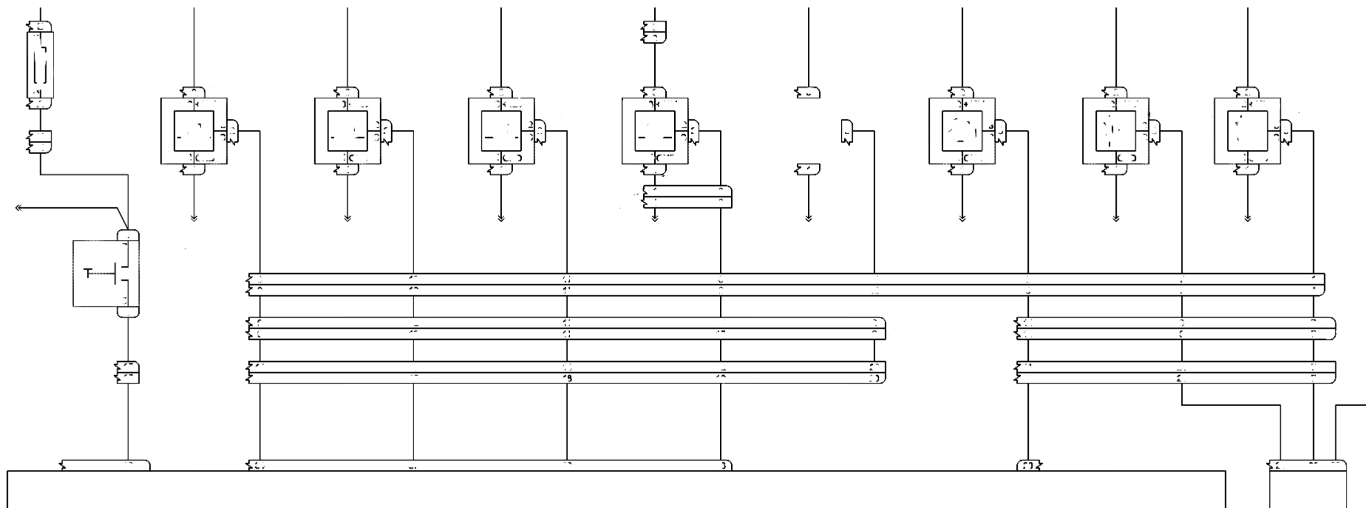
- Imagine you own a machine manufacturing company.
- The company produces various models with different equipment and individual custom-made parts.
- In the case of a malfunction, a technician needs a drawn cable plan to understand the particular interdependencies.
- Hand-drawn plans for all variants are unreasonably expensive.

$\Rightarrow$ draw plans automatically s.t. they resemble hand-drawn plans

Motivation

- Imagine you own a machine manufacturing company.
- The company produces various models with different equipment and individual custom-made parts.
- In the case of a malfunction, a technician needs a drawn cable plan to understand the particular interdependencies.
- Hand-drawn plans for all variants are unreasonably expensive.

$\Rightarrow$ draw plans automatically s.t. they resemble hand-drawn plans

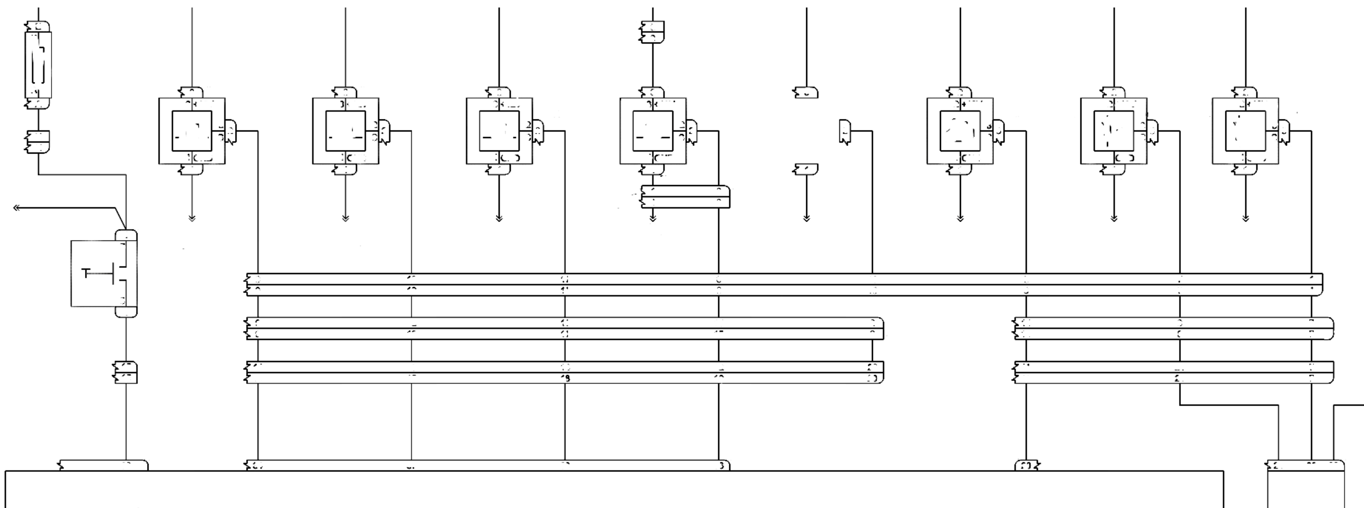


Motivation

- Imagine you own a machine manufacturing company.
- The company produces various models with different equipment and individual custom-made parts.
- In the case of a malfunction, a technician needs a drawn cable plan to understand the particular interdependencies.
- Hand-drawn plans for all variants are unreasonably expensive.

$\Rightarrow$ draw plans automatically s.t. they resemble hand-drawn plans

- orthogonal style
- vertices arranged on few layers

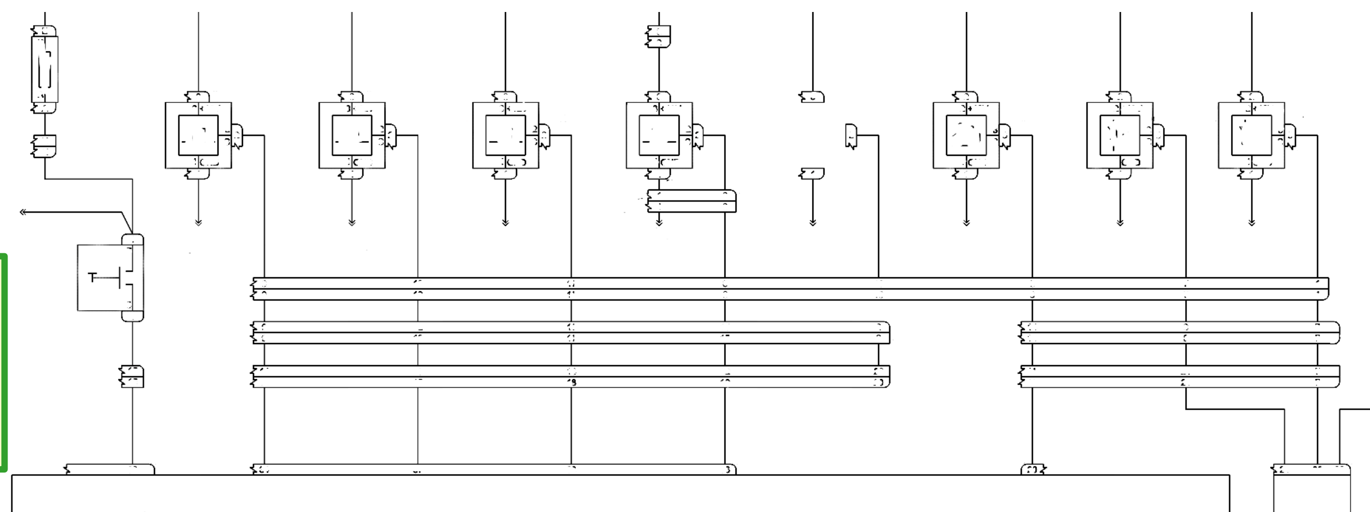


Motivation

- Imagine you own a machine manufacturing company.
- The company produces various models with different equipment and individual custom-made parts.
- In the case of a malfunction, a technician needs a drawn cable plan to understand the particular interdependencies.
- Hand-drawn plans for all variants are unreasonably expensive.

$\Rightarrow$ draw plans automatically s.t. they resemble hand-drawn plans

- orthogonal style
- vertices arranged on few layers



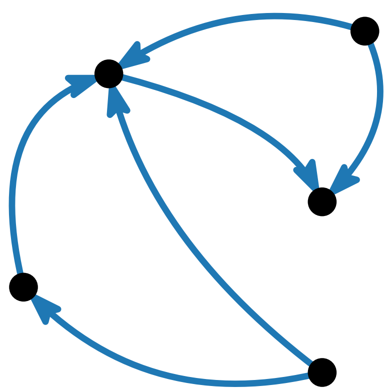
$\Rightarrow$ **use layered graph drawing algorithm**

Introduction: Layered Graph Drawing

- Given: directed acyclic graph $G = (V, A)$

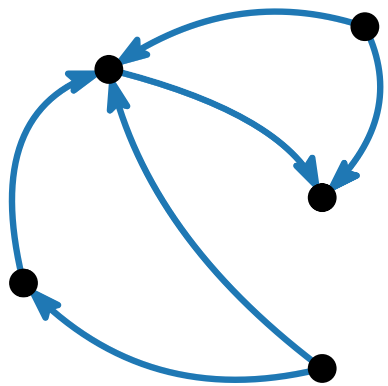
Introduction: Layered Graph Drawing

- Given: directed acyclic graph $G = (V, A)$



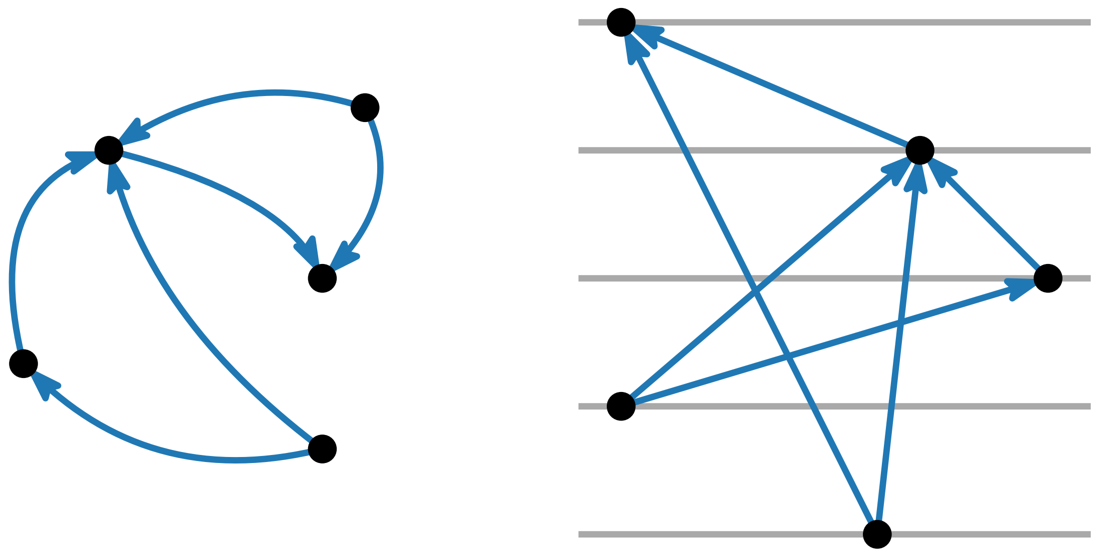
Introduction: Layered Graph Drawing

- Given: directed acyclic graph $G = (V, A)$
- Task: place vertices onto distinct horizontal lines (*layers*) s.t. all edges are directed upwards



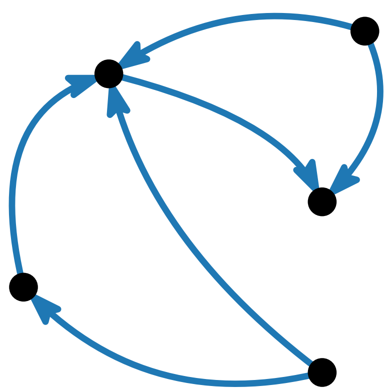
Introduction: Layered Graph Drawing

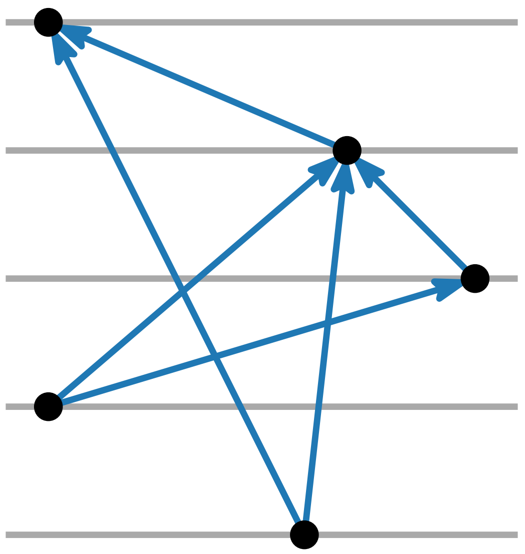
- Given: directed acyclic graph $G = (V, A)$
- Task: place vertices onto distinct horizontal lines (*layers*)
  s.t. all edges are directed upwards



Introduction: Layered Graph Drawing

- Given: directed acyclic graph $G = (V, A)$
- Task: place vertices onto distinct horizontal lines (*layers*)
  s.t. all edges are directed upwards

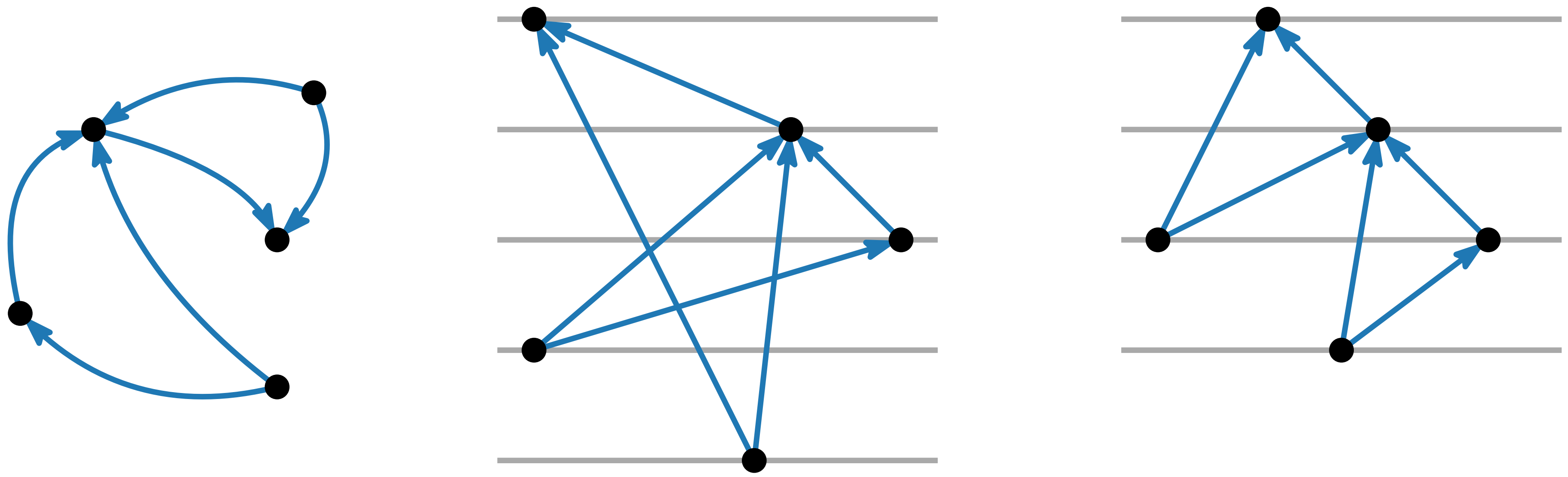




- Goals motivated by graph drawing aesthetics:
  few crossings, few layers, good aspect ratio, ...

Introduction: Layered Graph Drawing

- Given: directed acyclic graph $G = (V, A)$
- Task: place vertices onto distinct horizontal lines (*layers*)
  s.t. all edges are directed upwards



- Goals motivated by graph drawing aesthetics:
  few crossings, few layers, good aspect ratio, ...

Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

Introduction: Sugiyama Framework (1981)

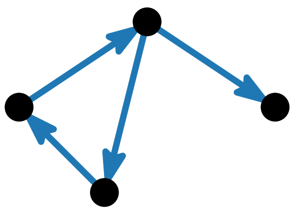
Consists of 5 phases:

1. cycle elimination

Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

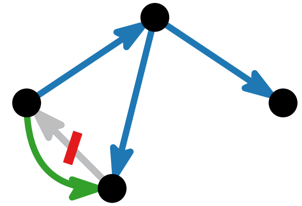
1. cycle elimination



Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

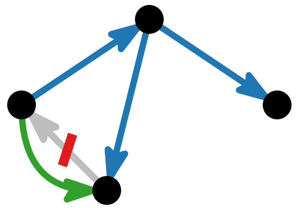
1. cycle elimination



Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

1. cycle elimination

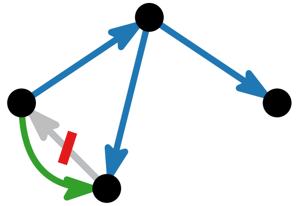


2. layer assignment

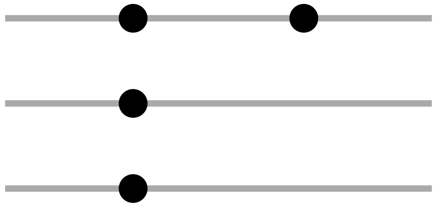
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

1. cycle elimination



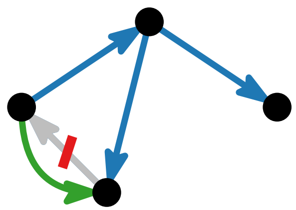
2. layer assignment



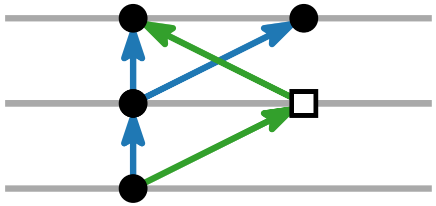
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

1. cycle elimination



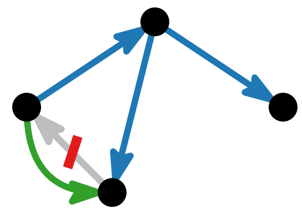
2. layer assignment



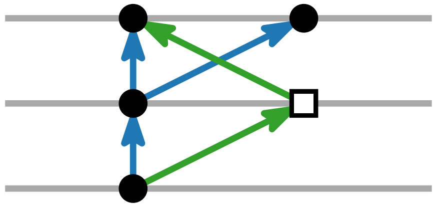
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

1. cycle elimination



2. layer assignment

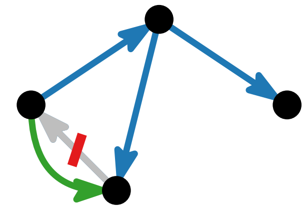


3. crossing minimization

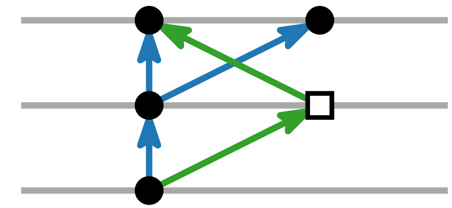
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

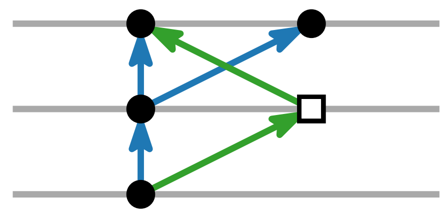
1. cycle elimination



2. layer assignment



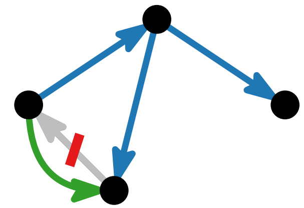
3. crossing minimization



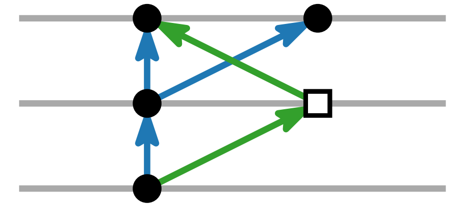
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

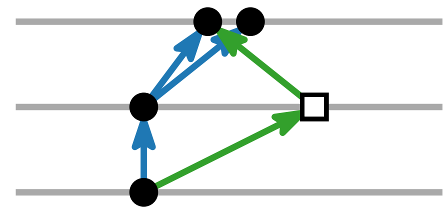
1. cycle elimination



2. layer assignment



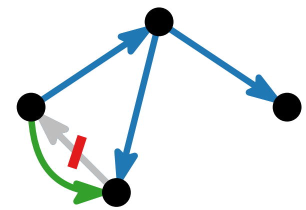
3. crossing minimization



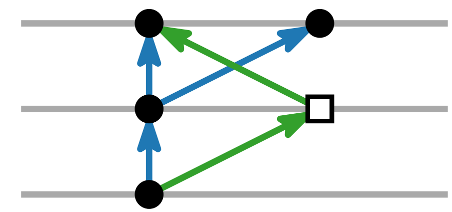
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

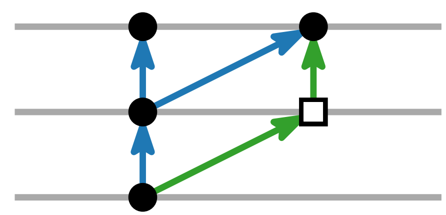
1. cycle elimination



2. layer assignment



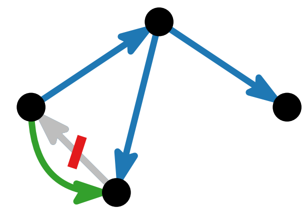
3. crossing minimization



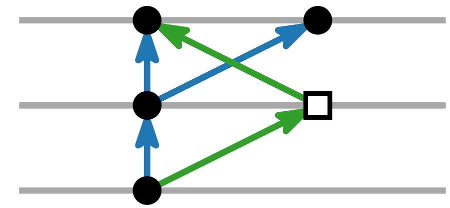
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

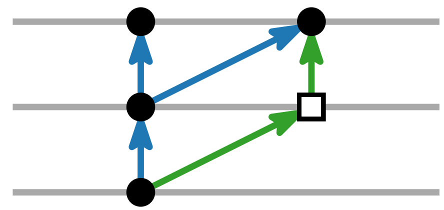
1. cycle elimination



2. layer assignment



3. crossing minimization

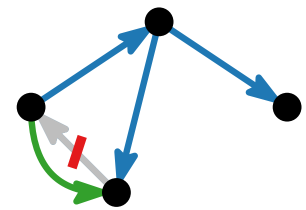


4. node placement

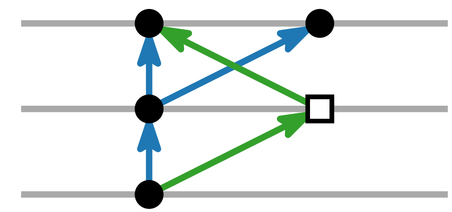
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

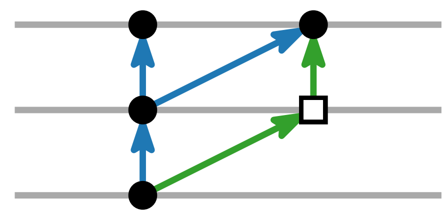
1. cycle elimination



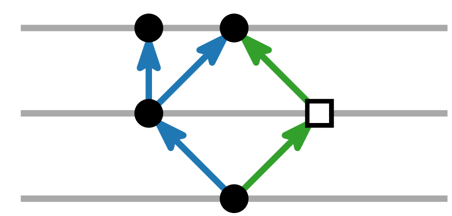
2. layer assignment



3. crossing minimization



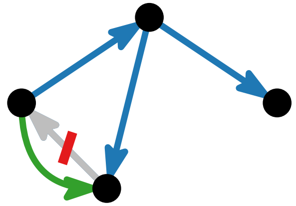
4. node placement



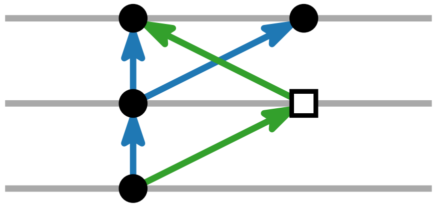
Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

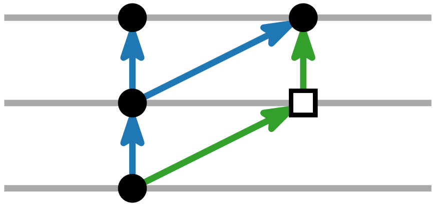
1. cycle elimination



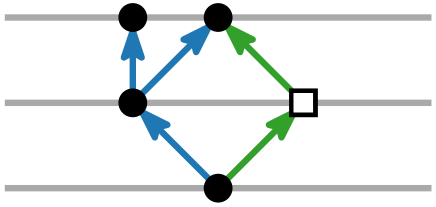
2. layer assignment



3. crossing minimization



4. node placement

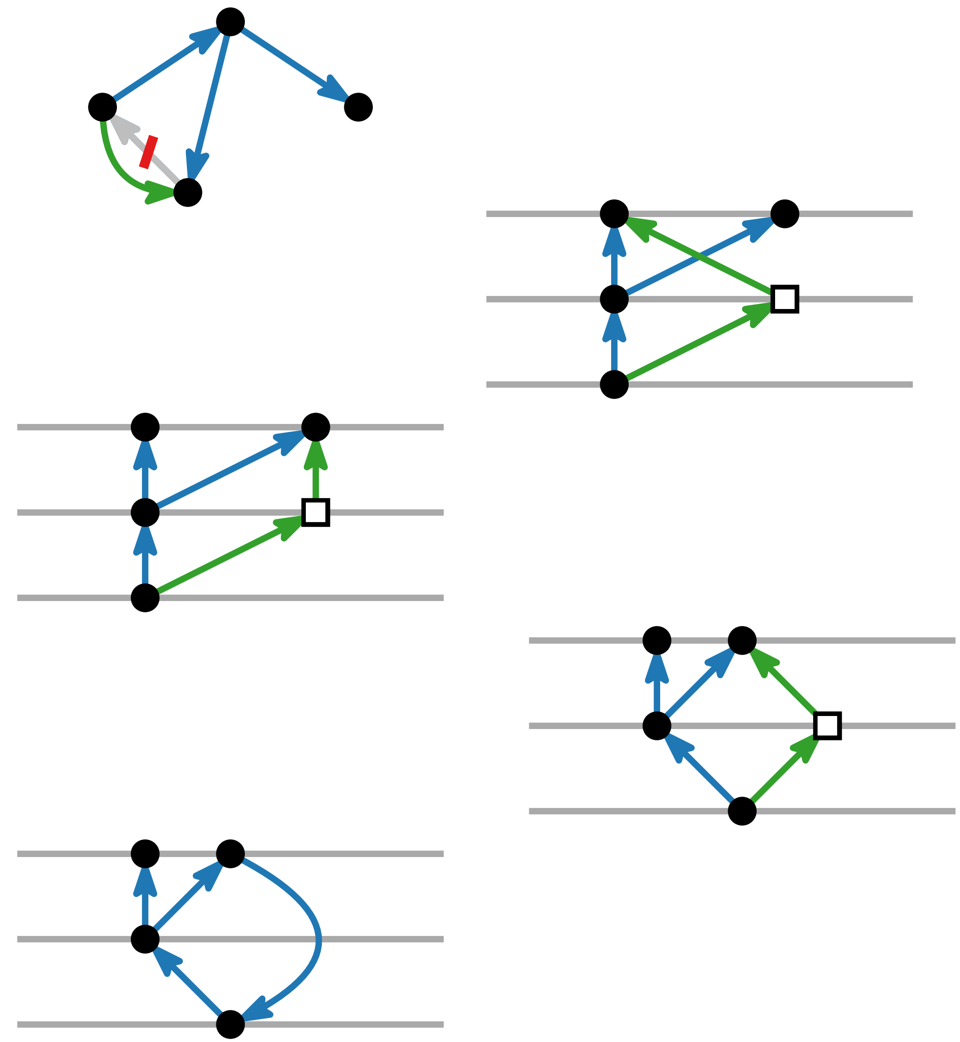


5. edge routing

Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

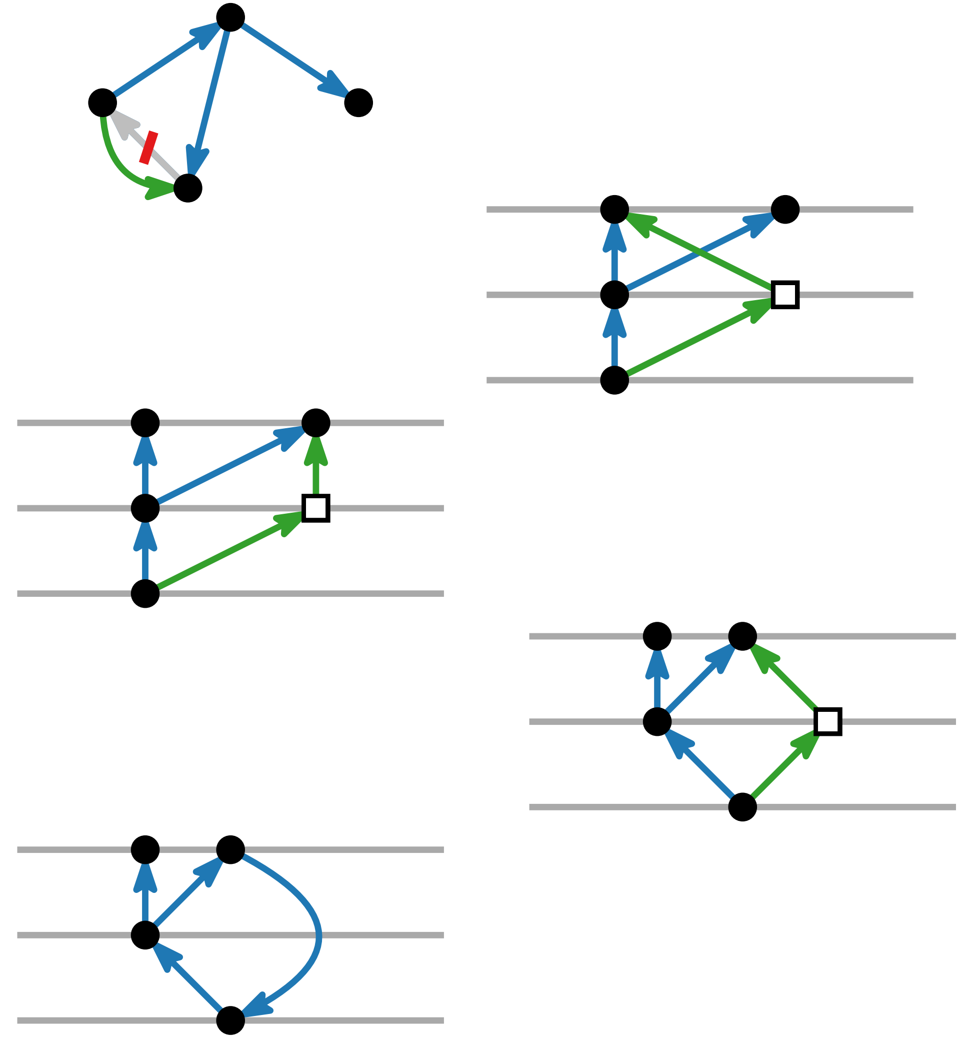
1. cycle elimination
2. layer assignment
3. crossing minimization
4. node placement
5. edge routing



Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

1. cycle elimination
2. layer assignment
3. crossing minimization
4. node placement
5. edge routing



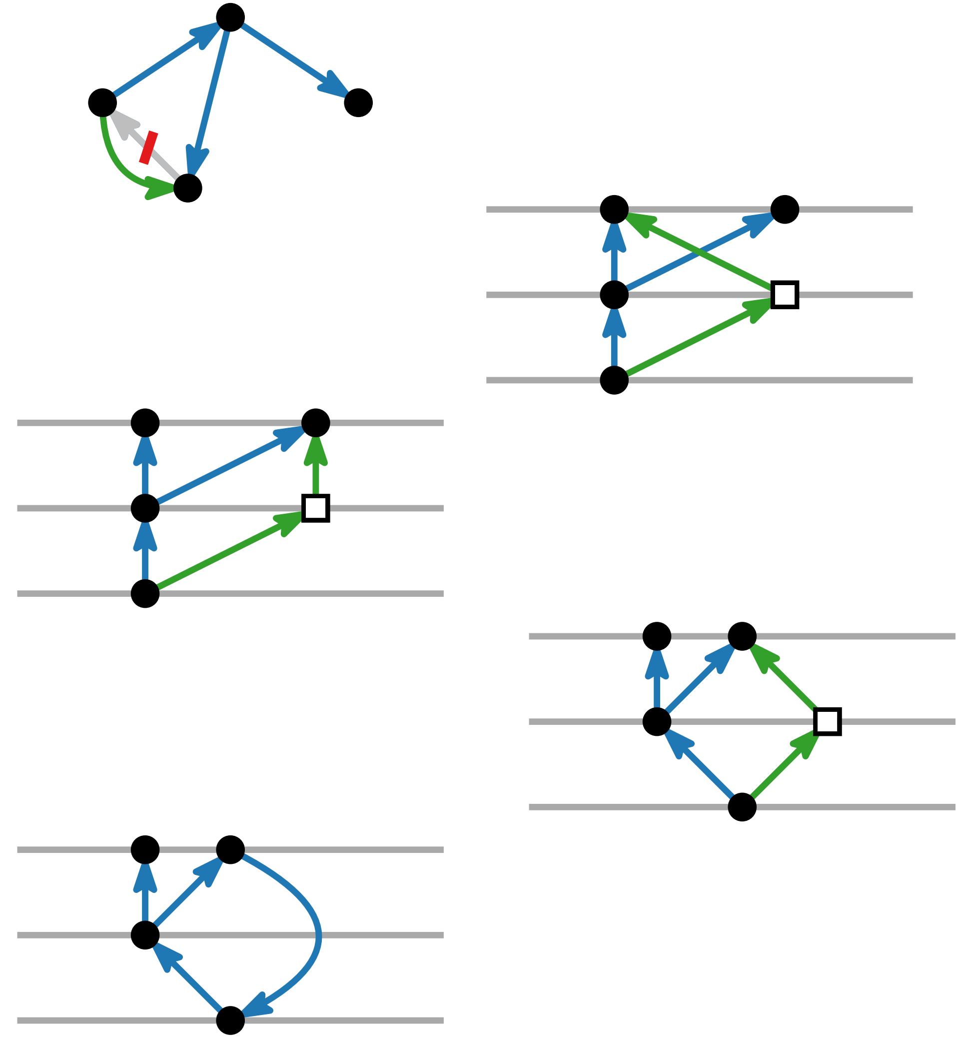
contains NP-hard tasks

Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

1. cycle elimination
2. layer assignment
   (for max. width)
3. crossing minimization
4. node placement
5. edge routing

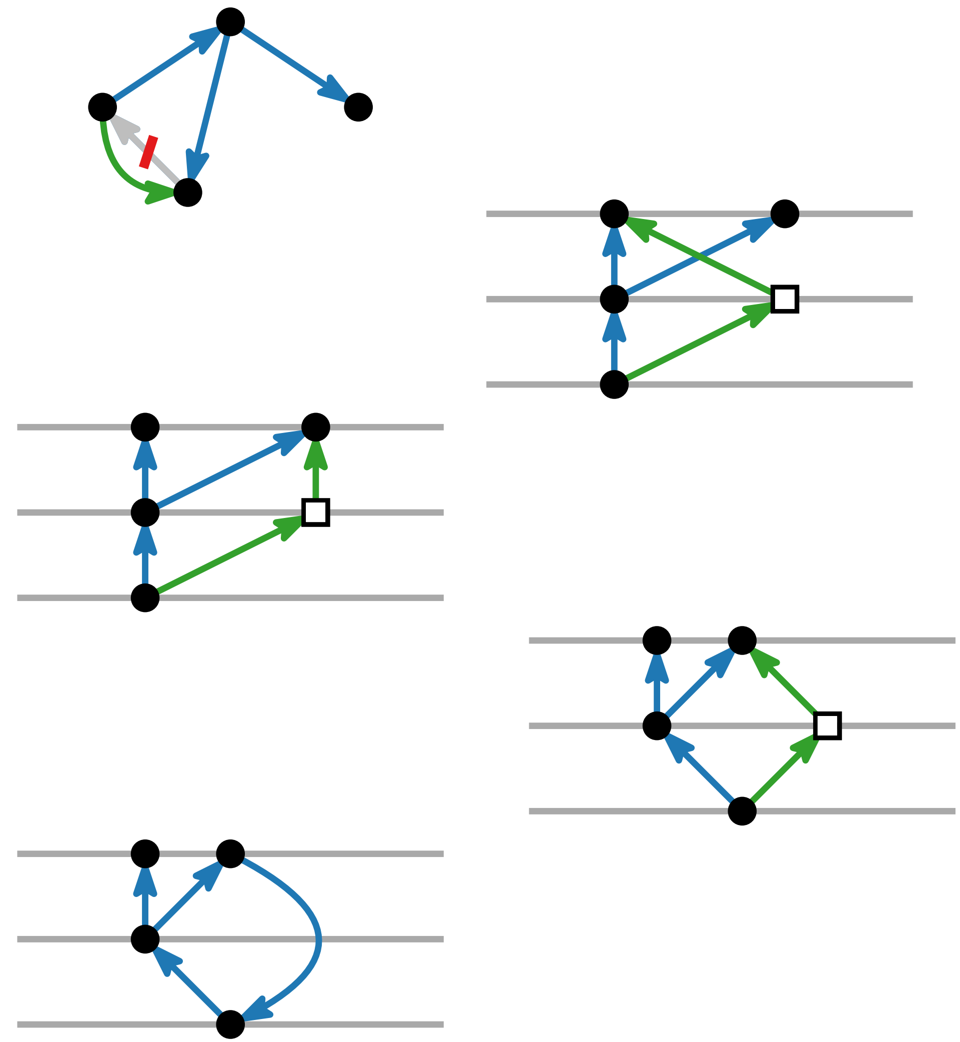
contains NP-hard tasks



Introduction: Sugiyama Framework (1981)

Consists of 5 phases:

1. cycle elimination
2. layer assignment
(for max. width)
3. crossing minimization
4. node placement
5. edge routing



contains NP-hard tasks

⇒ use heuristics

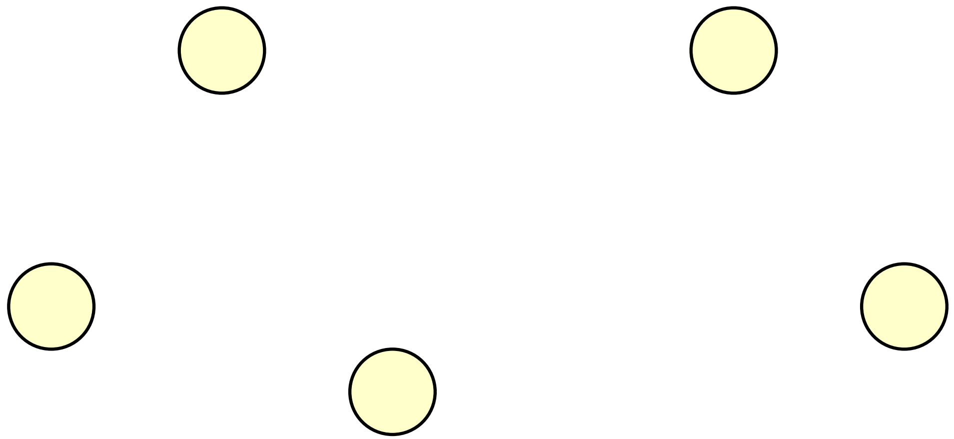
Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

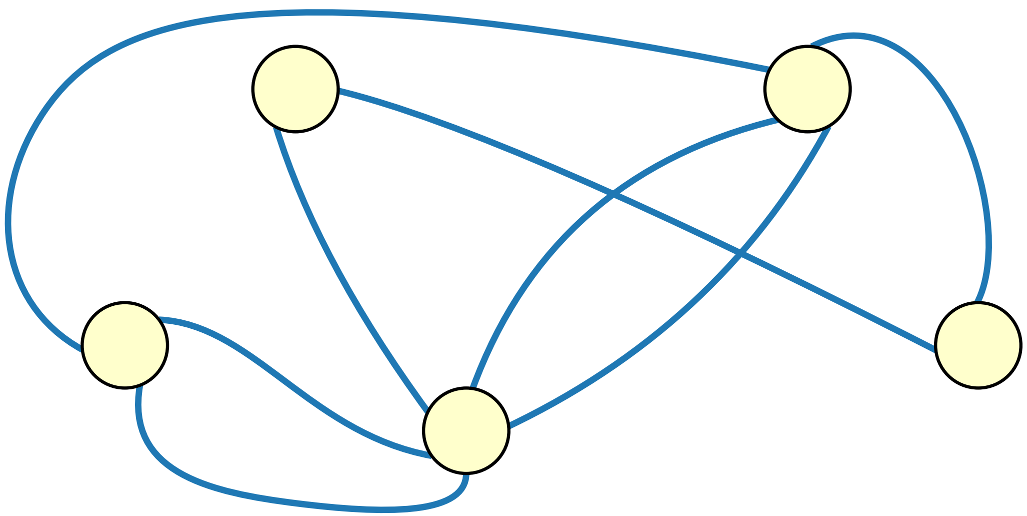
- vertex set V



Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

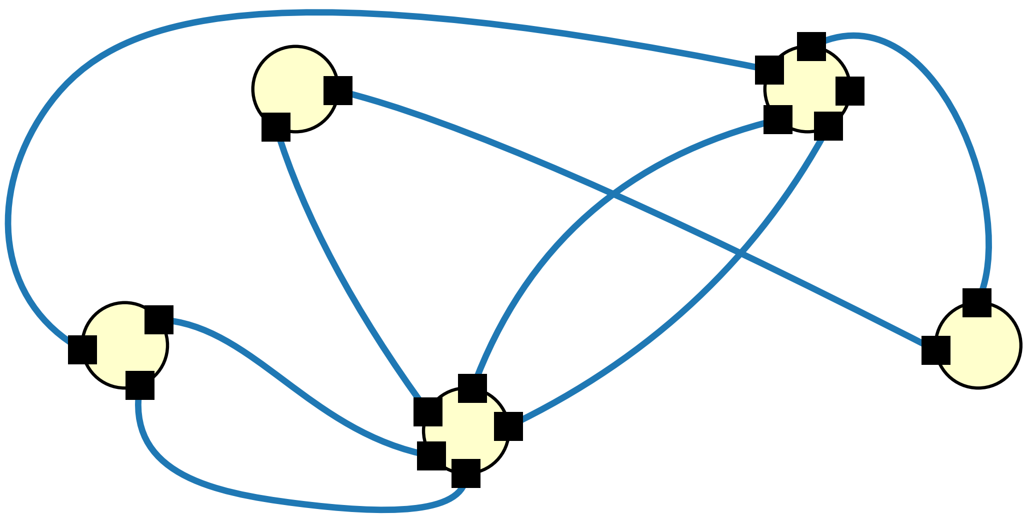
- vertex set V
- edge set E



Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

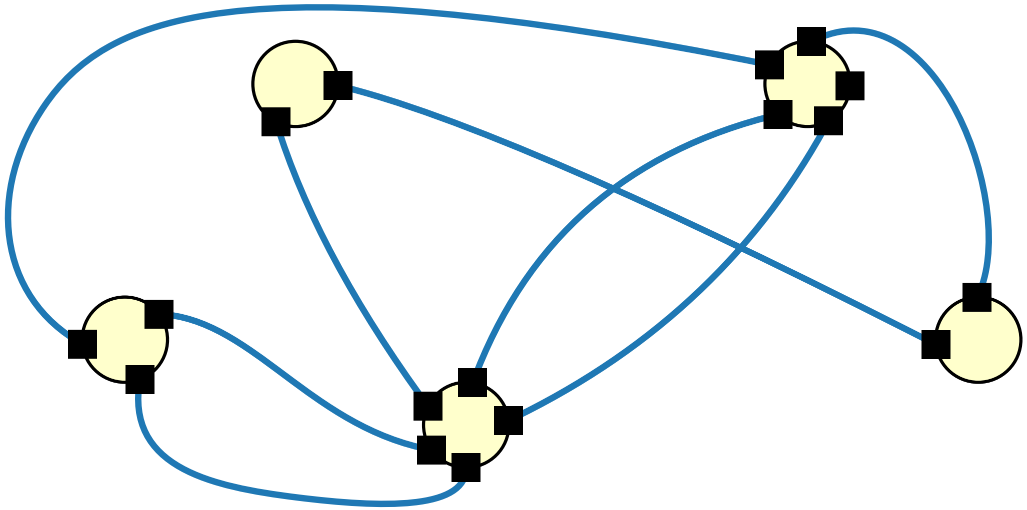
- vertex set V
- edge set E
- port set P s.t. each $p \in P$ belongs to some $v \in V$



Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

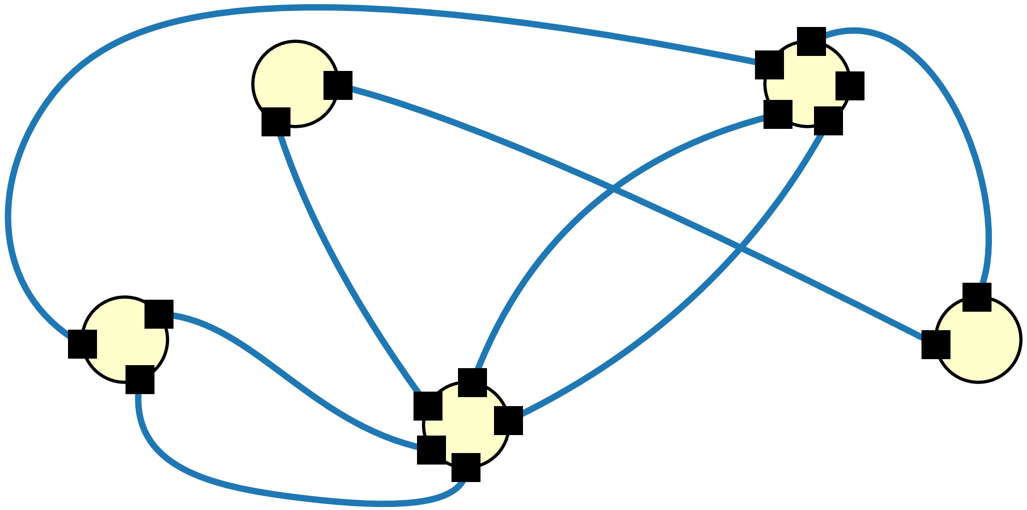
- vertex set V
- edge set $E \ni e \colon \{p_1, p_2\} \in \binom{P}{2}$
- port set P s.t. each $p \in P$ belongs to some $v \in V$



Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

- vertex set V
- edge set $E \ni e\colon \{p_1, p_2\} \in \binom{P}{2}$
- port set P s.t. each $p \in P$ belongs to some $v \in V$

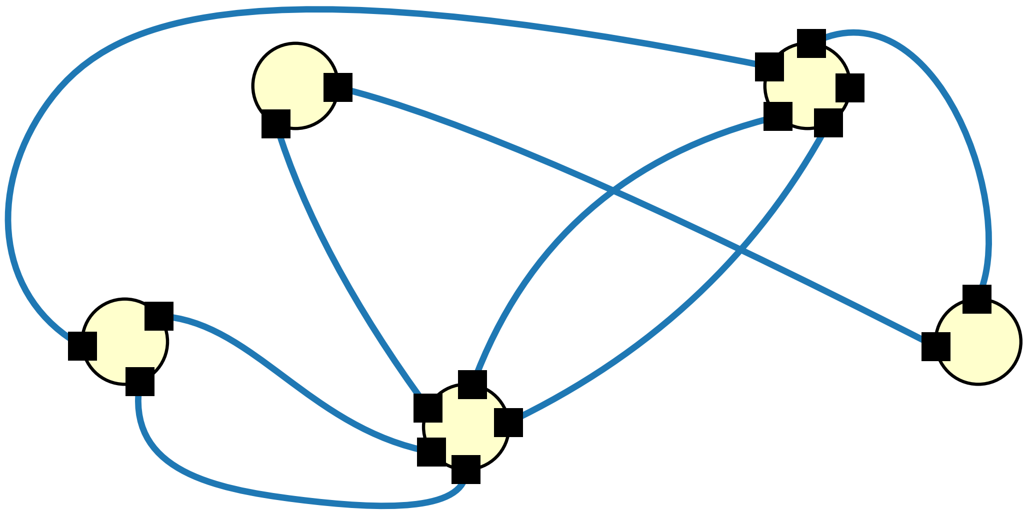


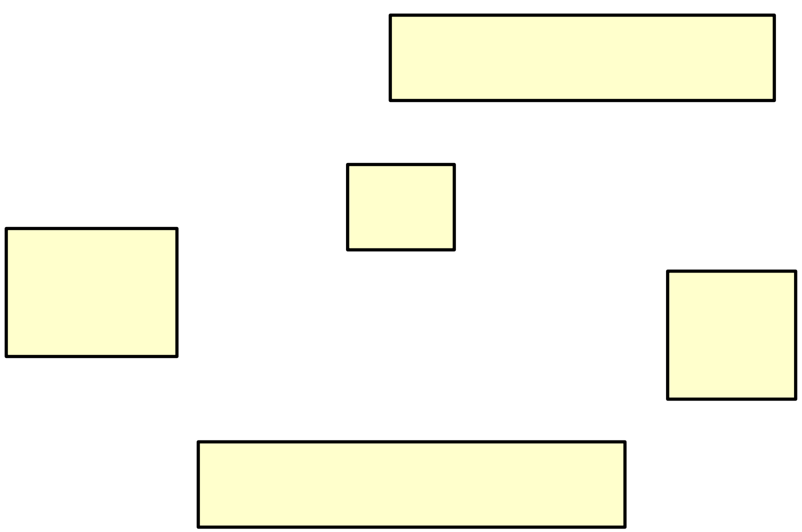
Orthogonal drawing style:

Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

- vertex set V
- edge set $E \ni e\colon \{p_1, p_2\} \in \binom{P}{2}$
- port set P s.t. each $p \in P$ belongs to some $v \in V$





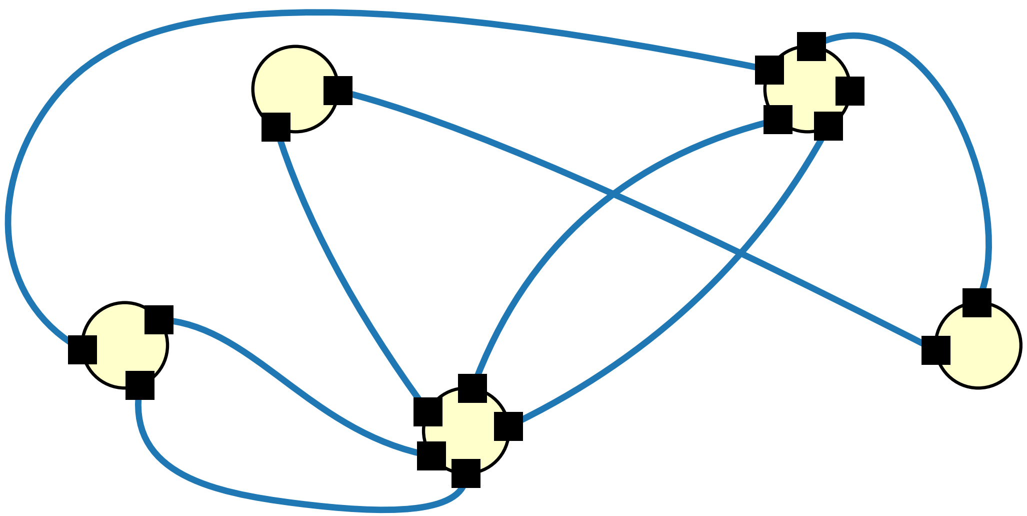
Orthogonal drawing style:

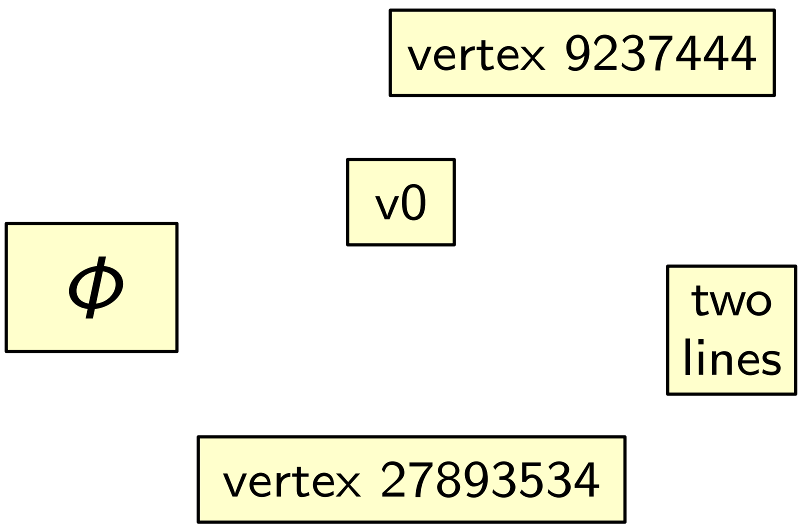
- $v \in V$: axis-aligned rectangle of width $\geq w(v)$, height $\geq h(v)$

Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

- vertex set V
- edge set $E \ni e$: $\{p_1, p_2\} \in \binom{P}{2}$
- port set P s.t. each $p \in P$ belongs to some $v \in V$





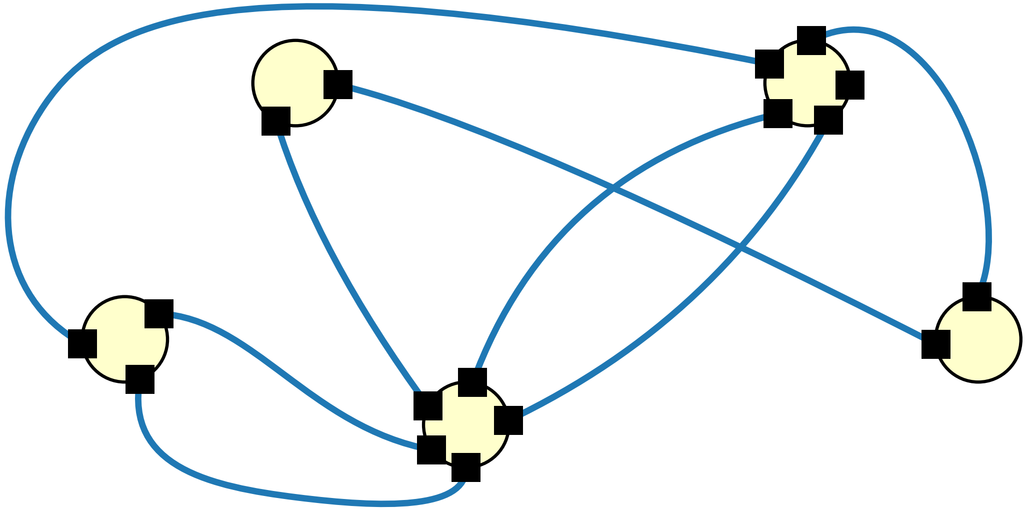
Orthogonal drawing style:

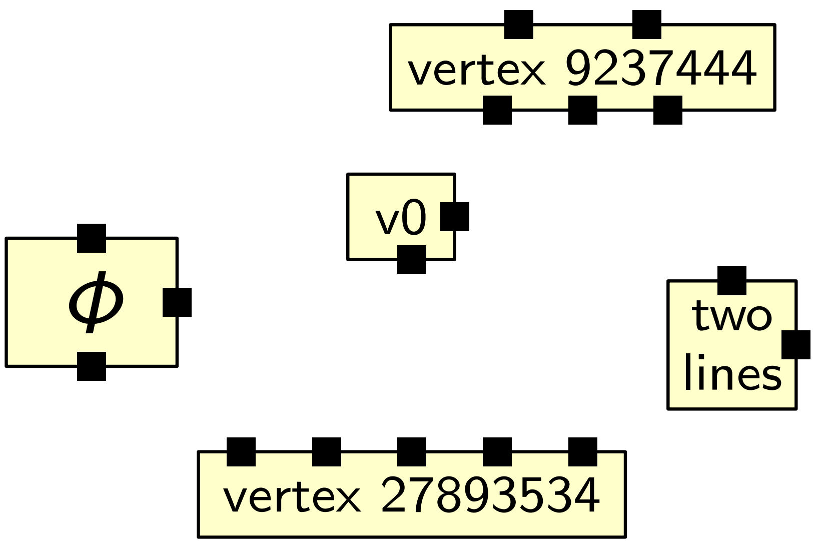
- $v \in V$: axis-aligned rectangle of width $\geq w(v)$, height $\geq h(v)$

Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

- vertex set V
- edge set $E \ni e$: $\{p_1, p_2\} \in \binom{P}{2}$
- port set P s.t. each $p \in P$ belongs to some $v \in V$





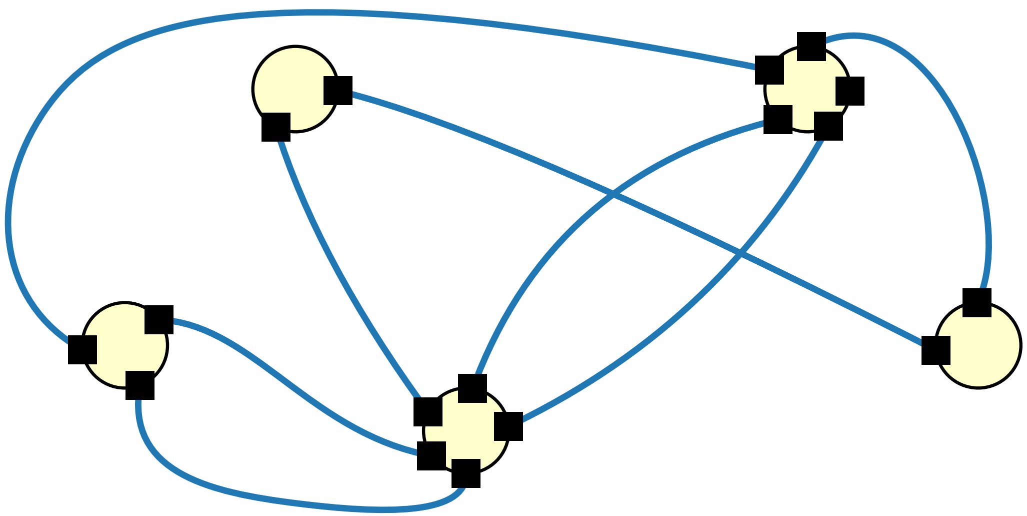
Orthogonal drawing style:

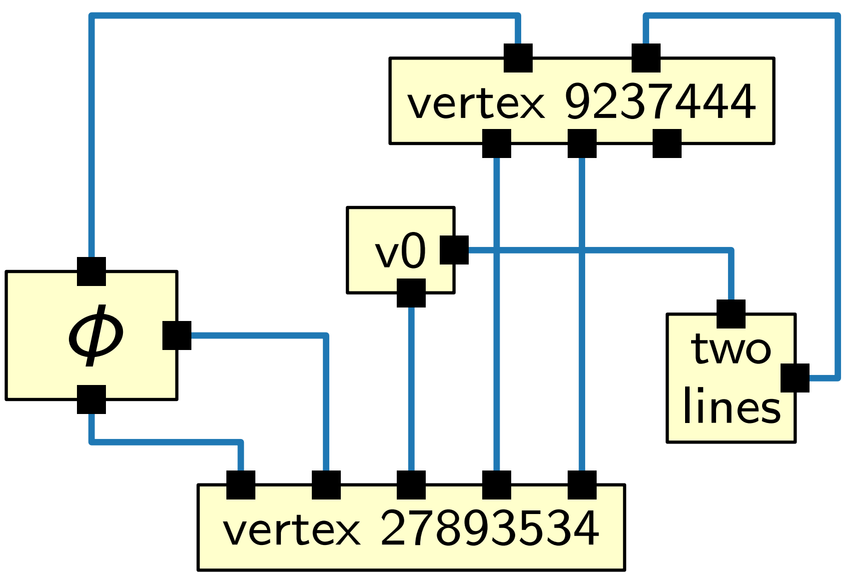
- $v \in V$: axis-aligned rectangle of width $\geq w(v)$, height $\geq h(v)$
- $p \in P$: small box on the boundary of its vertex

Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:

- vertex set V
- edge set $E \ni e \colon \{p_1, p_2\} \in \binom{P}{2}$
- port set P s.t. each $p \in P$ belongs to some $v \in V$





Orthogonal drawing style:

- $v \in V$: axis-aligned rectangle of width $\geq w(v)$, height $\geq h(v)$
- $p \in P$: small box on the boundary of its vertex
- $e \in E$: polyline of horizontal & vertical line segments

Previous Work

- Spönemann et al. (2009, 2014) consider graph drawing with port constraints in the Sugiyama framework.

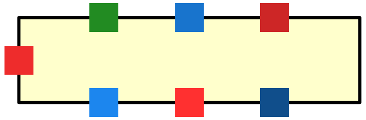
Previous Work

- Spönemann et al. (2009, 2014) consider graph drawing with port constraints in the Sugiyama framework.
- 4 levels of port constraints (assignment per vertex):

Previous Work

- Spönemann et al. (2009, 2014) consider graph drawing with port constraints in the Sugiyama framework.

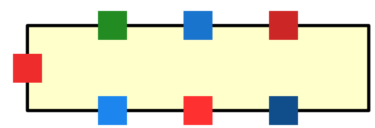
- 4 levels of port constraints (assignment per vertex):
  - Free



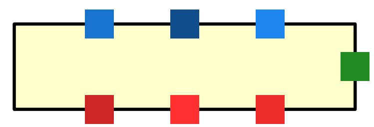
Previous Work

- Spönemann et al. (2009, 2014) consider graph drawing with port constraints in the Sugiyama framework.

- 4 levels of port constraints (assignment per vertex):
  - FREE

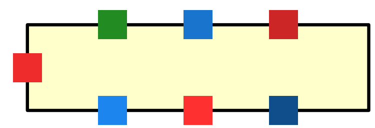
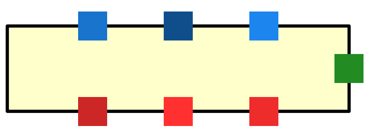
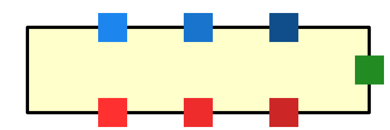


  - FIXEDSIDE



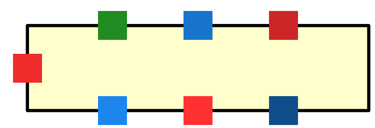
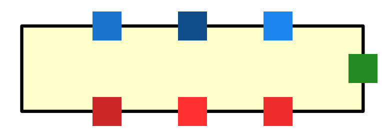
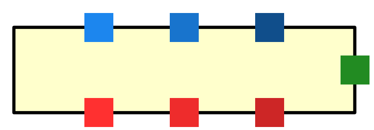
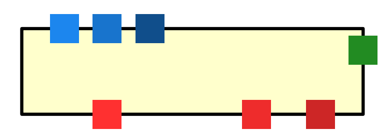
Previous Work

- Spönemann et al. (2009, 2014) consider graph drawing with port constraints in the Sugiyama framework.

- 4 levels of port constraints (assignment per vertex):
  - Free 
  - FixedSide 
  - FixedOrder 

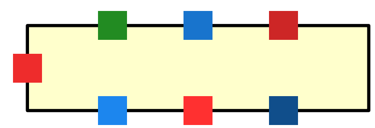
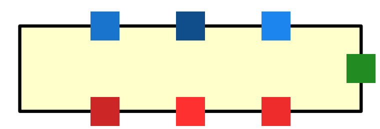
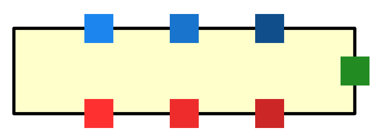
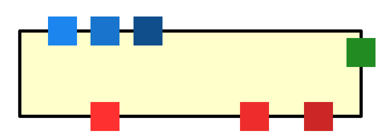
Previous Work

- Spönemann et al. (2009, 2014) consider graph drawing with port constraints in the Sugiyama framework.

- 4 levels of port constraints (assignment per vertex):
  - FREE 
  - FIXEDSIDE 
  - FIXEDORDER 
  - FIXEDPOS 

Previous Work

- Spönemann et al. (2009, 2014) consider graph drawing with port constraints in the Sugiyama framework.

- 4 levels of port constraints (assignment per vertex):
  - Free 
  - FixedSide 
  - FixedOrder 
  - FixedPos 

- Open source implemenation in Java as *KIELER* (later: *eclipse.elk*) available

Contribution

We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:

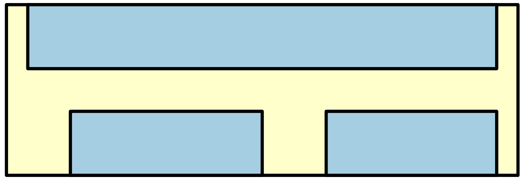
Contribution

We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:

- Port groups (generalizing port constraints FREE, FIXEDSIDE, FIXEDORDER)

Contribution

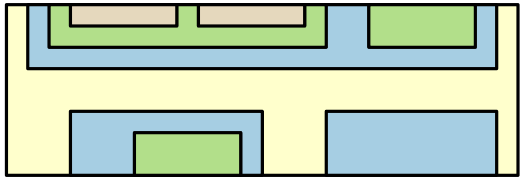
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints FREE, FIXEDSIDE, FIXEDORDER)
  - can be assigend to a vertex side or FREE

Contribution

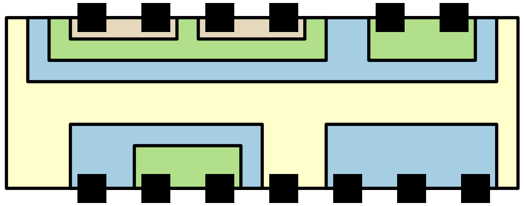
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints Free, FixedSide, FixedOrder)
  - can be assigend to a vertex side or Free
  - can be nested

Contribution

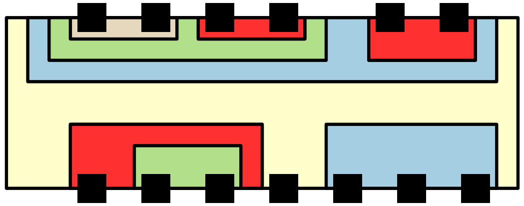
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints FREE, FIXEDSIDE, FIXEDORDER)
  - can be assigend to a vertex side or FREE
  - can be nested

Contribution

We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:

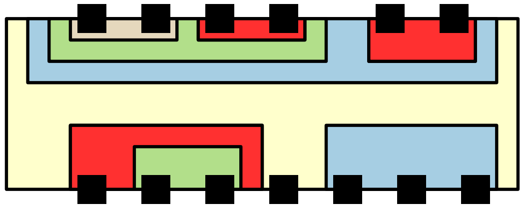


- Port groups (generalizing port constraints Free, FixedSide, FixedOrder)
  - can be assigend to a vertex side or Free
  - can be nested
  - internal order can be fixed or variable

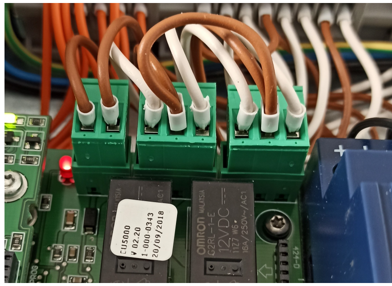
 

Contribution

We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:

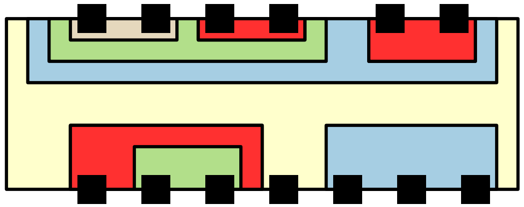


- Port groups (generalizing port constraints FREE, FIXEDSIDE, FIXEDORDER)
  - can be assigend to a vertex side or FREE
  - can be nested
  - internal order can be fixed or variable

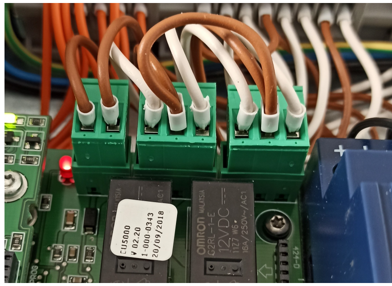


Contribution

We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



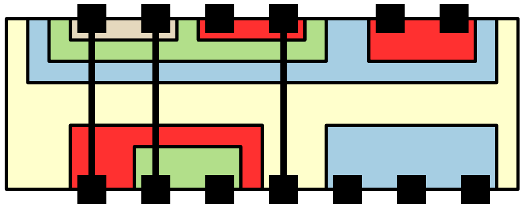
- Port groups (generalizing port constraints Free, FixedSide, FixedOrder)
  - can be assigend to a vertex side or Free
  - can be nested
  - internal order can be fixed or variable



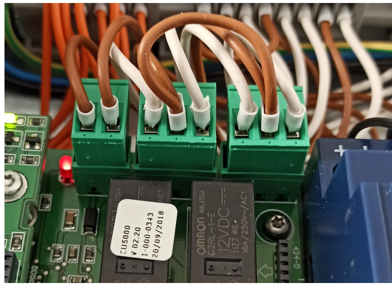
- Port pairings: require 2 ports to be on an axis-parallel line

Contribution

We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



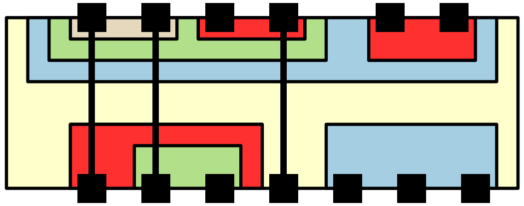
- Port groups (generalizing port constraints Free, FixedSide, FixedOrder)



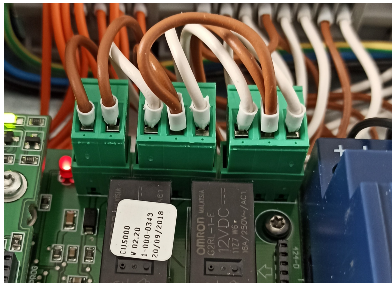
  - can be assigend to a vertex side or Free
  - can be nested
  - internal order can be fixed or variable
- Port pairings: require 2 ports to be on an axis-parallel line

Contribution

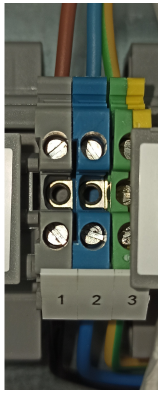
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints FREE, FIXEDSIDE, FIXEDORDER)

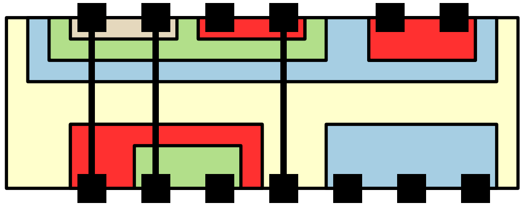


  - can be assigend to a vertex side or FREE
  - can be nested
  - internal order can be fixed or variable
- Port pairings: require 2 ports to be on an axis-parallel line

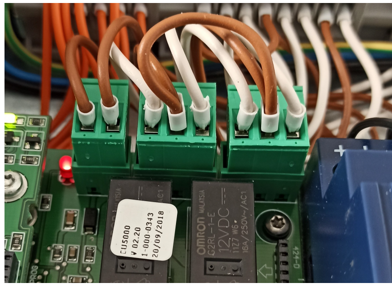


Contribution

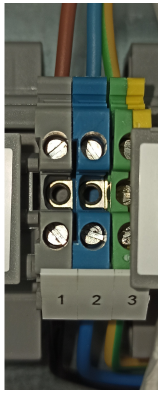
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints FREE, FIXEDSIDE, FIXEDORDER)



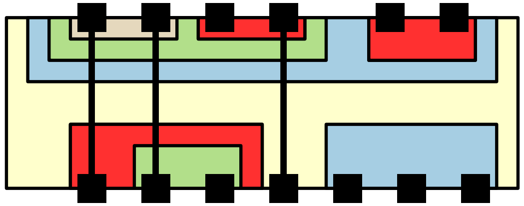
  - can be assigend to a vertex side or FREE
  - can be nested
  - internal order can be fixed or variable
- Port pairings: require 2 ports to be on an axis-parallel line



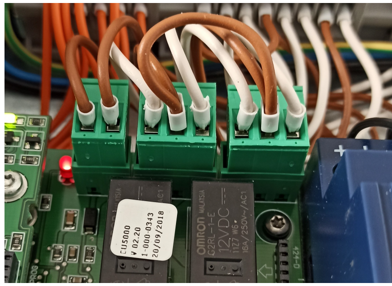
- Draw undirected graphs by orienting the edges using

Contribution

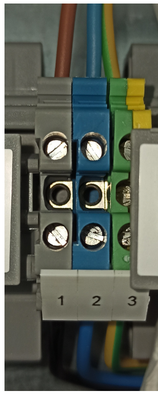
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints Free, FixedSide, FixedOrder)
  - can be assigend to a vertex side or Free
  - can be nested
  - internal order can be fixed or variable

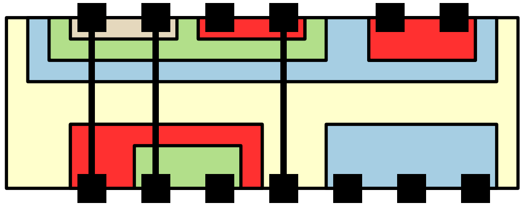


- Port pairings: require 2 ports to be on an axis-parallel line
- Draw undirected graphs by orienting the edges using
  - breadth-first search (orient in direction of discovery)

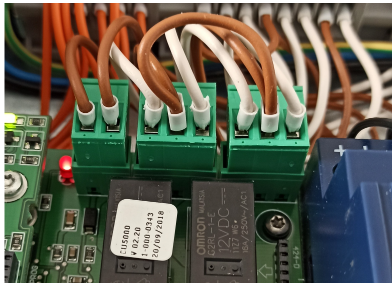


Contribution

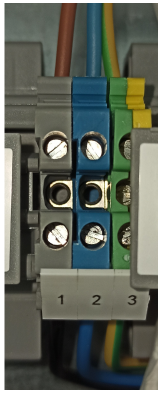
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints Free, FixedSide, FixedOrder)
  - can be assigend to a vertex side or Free
  - can be nested
  - internal order can be fixed or variable



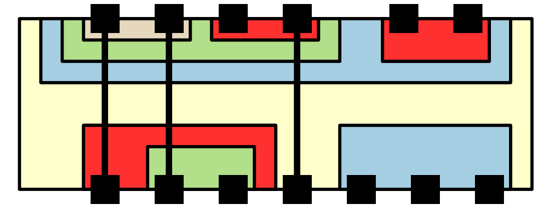
- Port pairings: require 2 ports to be on an axis-parallel line



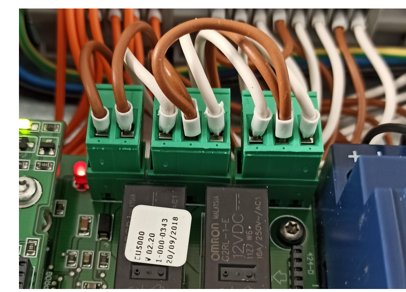
- Draw undirected graphs by orienting the edges using
  - breadth-first search (orient in direction of discovery)
  - force-directed algorithm (orient all edges upwards)

Contribution

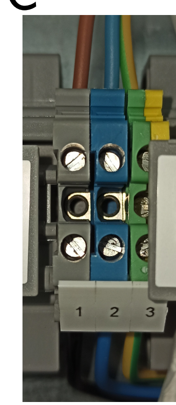
We extend an orthogonal-style layered graph drawing algorithm built on the Sugiyama framework with ports by:



- Port groups (generalizing port constraints Free, FixedSide, FixedOrder)



  - can be assigend to a vertex side or Free
  - can be nested
  - internal order can be fixed or variable
- Port pairings: require 2 ports to be on an axis-parallel line



- Draw undirected graphs by orienting the edges using
  - breadth-first search (orient in direction of discovery)
  - force-directed algorithm (orient all edges upwards)

We experimentally evaluate our variants on real cable plans and pseudo plans (we describe how we generate them from real data)

Our Extensions to the Sugiyama Framework

1. cycle elimination
2. layer assignment
3. crossing minimization
4. node placement
5. edge routing

Our Extensions to the Sugiyama Framework

1. ~~cycle elimination~~
   1. Orient undirected edges (w/o creating cycles)
      - with breadth-first search (BFS)
      - with force-directed algorithm (FD)
      - by random placement (RAND)
2. layer assignment
3. crossing minimization
4. node placement
5. edge routing

Our Extensions to the Sugiyama Framework

1. ~~cycle elimination~~
2. layer assignment
3. crossing minimization
4. node placement
5. edge routing

1. Orient undirected edges (w/o creating cycles)
   - with breadth-first search (BFS)
   - with force-directed algorithm (FD)
   - by random placement (RAND)

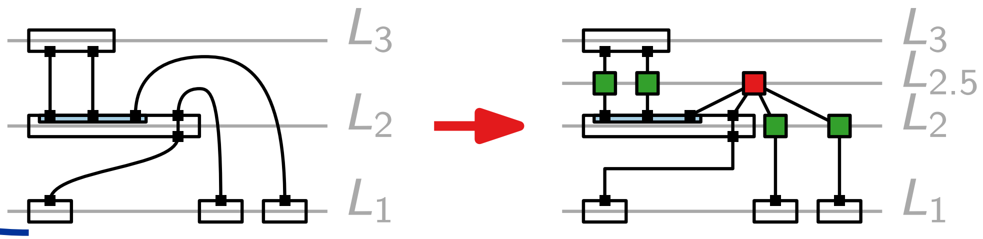
2.5 Orient ports acc. to port groups, insert *turning dummy vertices* for ports on the "wrong" side:

Our Extensions to the Sugiyama Framework

1. ~~cycle elimination~~
2. layer assignment
3. crossing minimization
4. node placement
5. edge routing

1. Orient undirected edges (w/o creating cycles)
   - with breadth-first search (BFS)
   - with force-directed algorithm (FD)
   - by random placement (RAND)

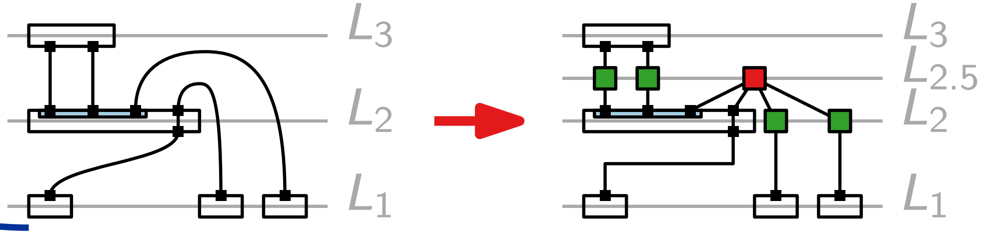
2.5 Orient ports acc. to port groups, insert *turning dummy vertices* for ports on the "wrong" side:



Our Extensions to the Sugiyama Framework

1. ~~cycle elimination~~
   1. Orient undirected edges (w/o creating cycles)
      - with breadth-first search (BFS)
      - with force-directed algorithm (FD)
      - by random placement (RAND)
2. layer assignment

   2.5 Orient ports acc. to port groups, insert *turning dummy vertices* for ports on the "wrong" side:



3. crossing minimization

   Well-established barycenter heuristic with respect to:
4. node placement
5. edge routing

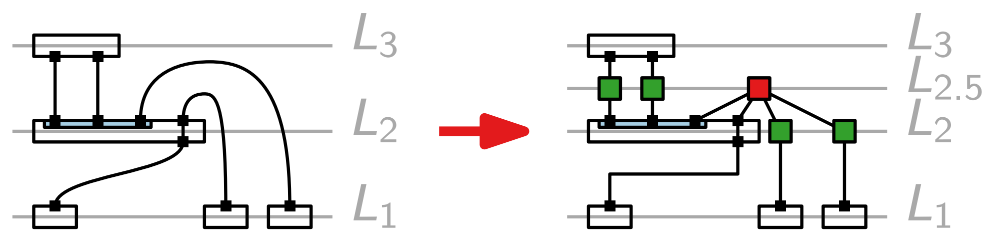
Our Extensions to the Sugiyama Framework

~~1. cycle elimination~~

1. Orient undirected edges (w/o creating cycles)
   - with breadth-first search (BFS)
   - with force-directed algorithm (FD)
   - by random placement (RAND)

2. layer assignment

2.5 Orient ports acc. to port groups, insert *turning dummy vertices* for ports on the "wrong" side:



3. crossing minimization

Well-established barycenter heuristic with respect to:

- VERTICES – sort ports afterwards
- PORTS – sort port groups & vtcs. recursively acc. to barycenters of their ports
- MIXED – for port pairings like PORTS, otherwise like VERTICES

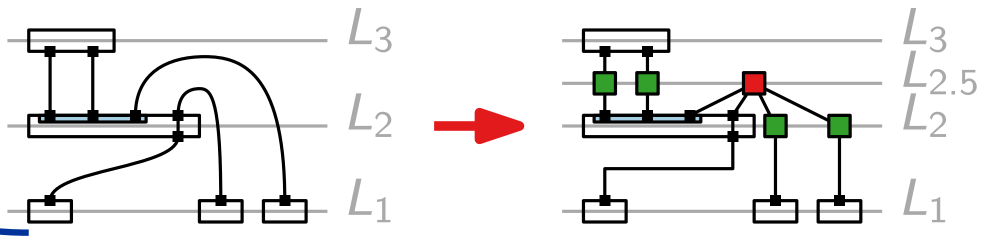
4. node placement

5. edge routing

Our Extensions to the Sugiyama Framework

1. ~~cycle elimination~~
   1. Orient undirected edges (w/o creating cycles)
      - with breadth-first search (BFS)
      - with force-directed algorithm (FD)
      - by random placement (RAND)
2. layer assignment

   2.5 Orient ports acc. to port groups, insert *turning dummy vertices* for ports on the “wrong” side:



3. crossing minimization

   Well-established barycenter heuristic with respect to:
   - VERTICES – sort ports afterwards
   - PORTS – sort port groups & vtcs. recursively acc. to barycenters of their ports
   - MIXED – for port pairings like PORTS, otherwise like VERTICES
4. node placement

   (Fixed) algorithm by Brandes & Köpf (GD’01)
5. edge routing

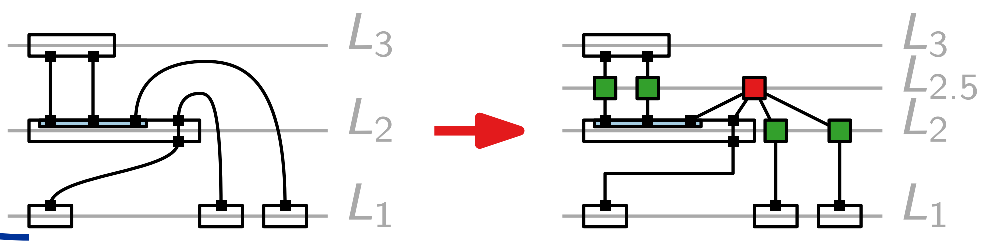
Our Extensions to the Sugiyama Framework

1. ~~cycle elimination~~

1. Orient undirected edges (w/o creating cycles)
   - with breadth-first search (BFS)
   - with force-directed algorithm (FD)
   - by random placement (RAND)

2. layer assignment

2.5 Orient ports acc. to port groups, insert *turning dummy vertices* for ports on the "wrong" side:



3. crossing minimization

Well-established barycenter heuristic with respect to:

- VERTICES – sort ports afterwards
- PORTS – sort port groups & vtcs. recursively acc. to barycenters of their ports
- MIXED – for port pairings like PORTS, otherwise like VERTICES

4. node placement

(Fixed) algorithm by Brandes & Köpf (GD'01)

5. edge routing

orthogonal

Experiments

- Real: 380 real cable plans of a large machine manufacturer

Experiments

- REAL: 380 real cable plans of a large machine manufacturer
- PSEUDO: 1139 cable plans artificially generated from REAL

Experiments

- REAL: 380 real cable plans of a large machine manufacturer
- PSEUDO: 1139 cable plans artificially generated from REAL
- Tested different variants of our algorithm:

Experiments

- Real: 380 real cable plans of a large machine manufacturer
- Pseudo: 1139 cable plans artificially generated from Real
- Tested different variants of our algorithm:
  - methods for orienting the edges

Experiments

- REAL: 380 real cable plans of a large machine manufacturer
- PSEUDO: 1139 cable plans artificially generated from REAL
- Tested different variants of our algorithm:
  - methods for orienting the edges
  - methods for crossing minimization

Experiments

- Real: 380 real cable plans of a large machine manufacturer
- Pseudo: 1139 cable plans artificially generated from Real
- Tested different variants of our algorithm:
  - methods for orienting the edges
  - methods for crossing minimization
- Measured #crossings, #bends of orthogonal output drawings

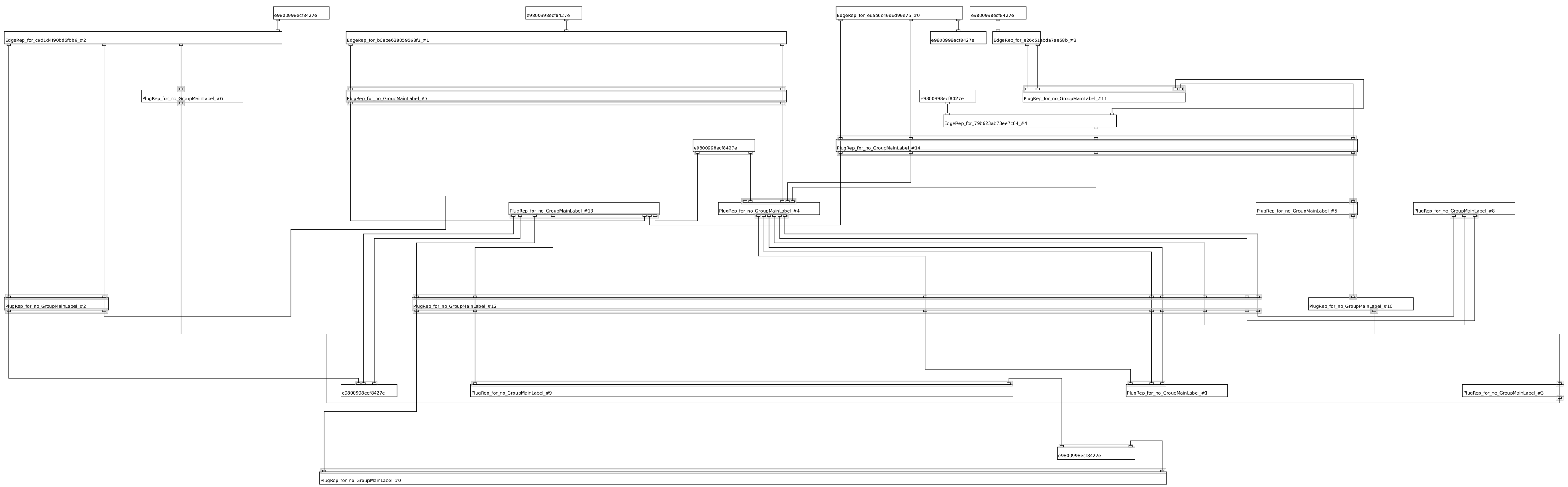
Experiments

- REAL: 380 real cable plans of a large machine manufacturer
- PSEUDO: 1139 cable plans artificially generated from REAL
- Tested different variants of our algorithm:
  - methods for orienting the edges
  - methods for crossing minimization
- Measured #crossings, #bends of orthogonal output drawings
- Took best of 5 executions for each plan & variant

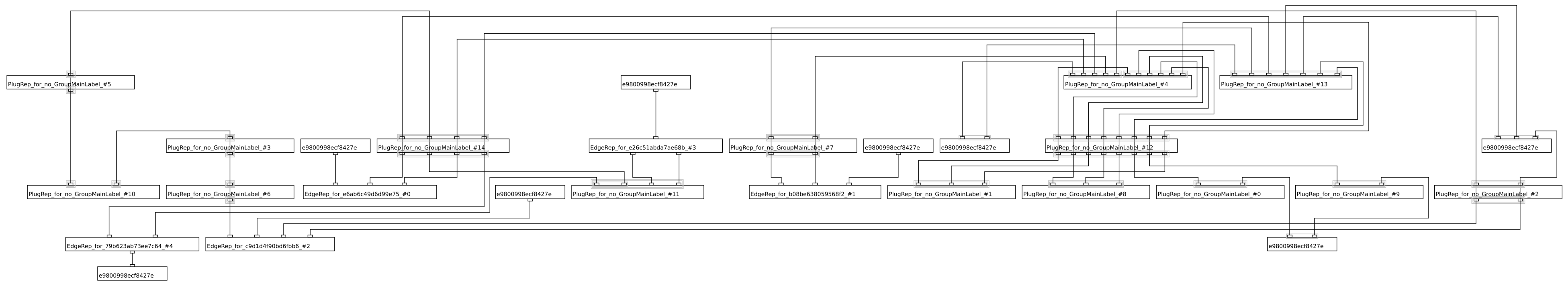
Experiments

- REAL: 380 real cable plans of a large machine manufacturer
- PSEUDO: 1139 cable plans artificially generated from REAL
- Tested different variants of our algorithm:
  - methods for orienting the edges
  - methods for crossing minimization
- Measured #crossings, #bends of orthogonal output drawings
- Took best of 5 executions for each plan & variant
- Our implementation in Java is available on github:
  `github.com/j-zink-wuerzburg`
  `.../praline`
  `.../pseudo-praline-plan-generation`

Example: (anonymized) plan from REAL



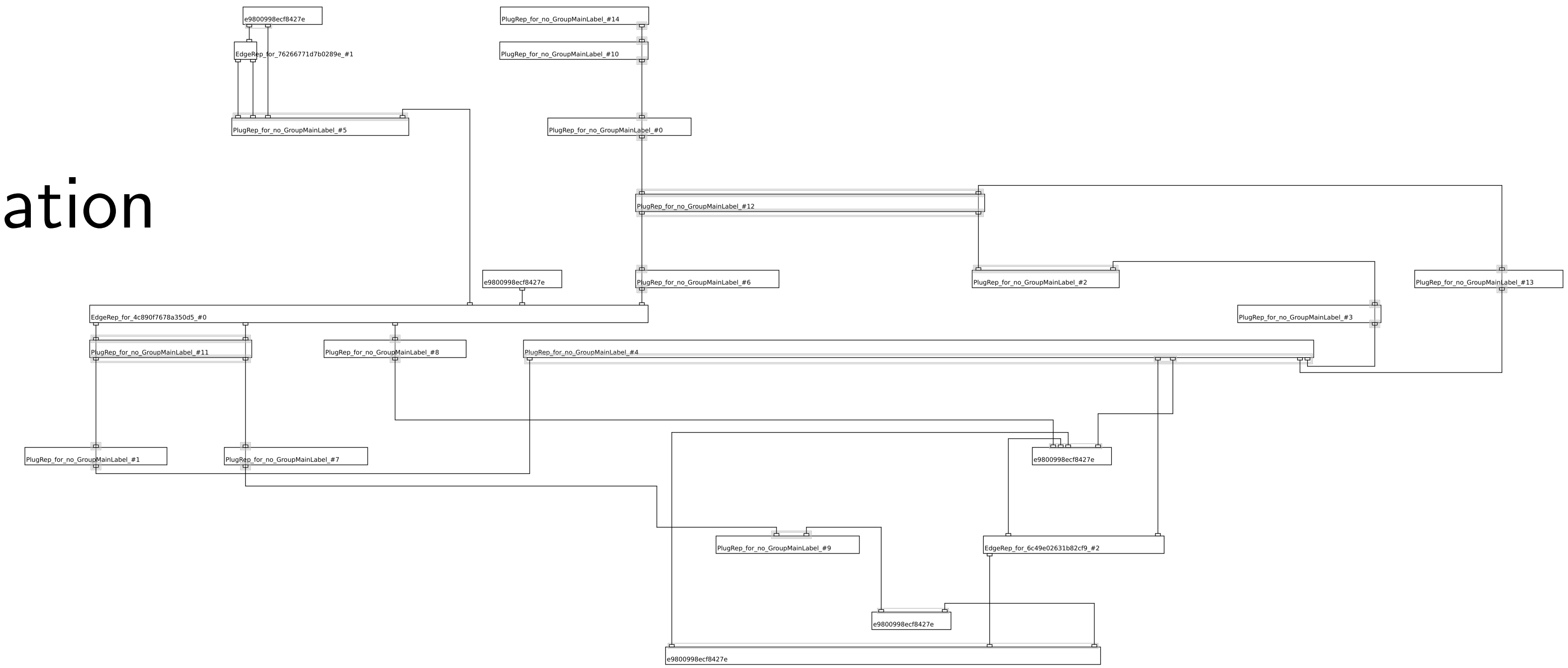
our implementation

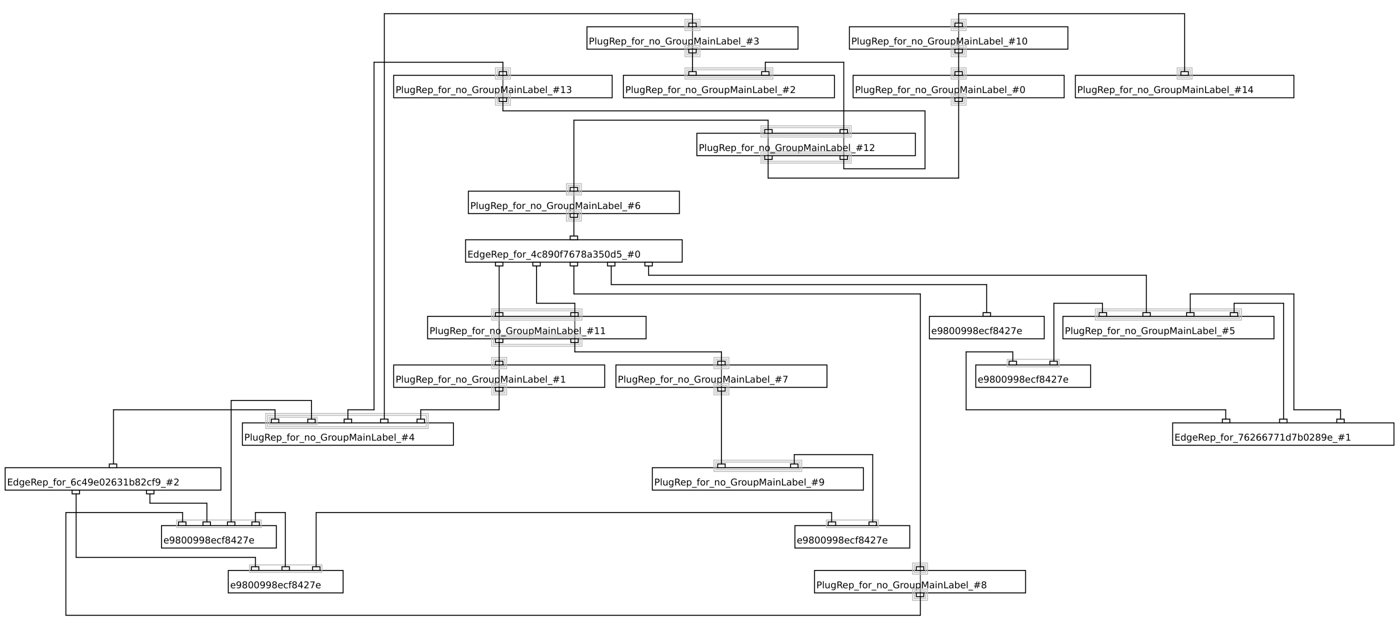


KIELER

Example: plan from PSEUDO

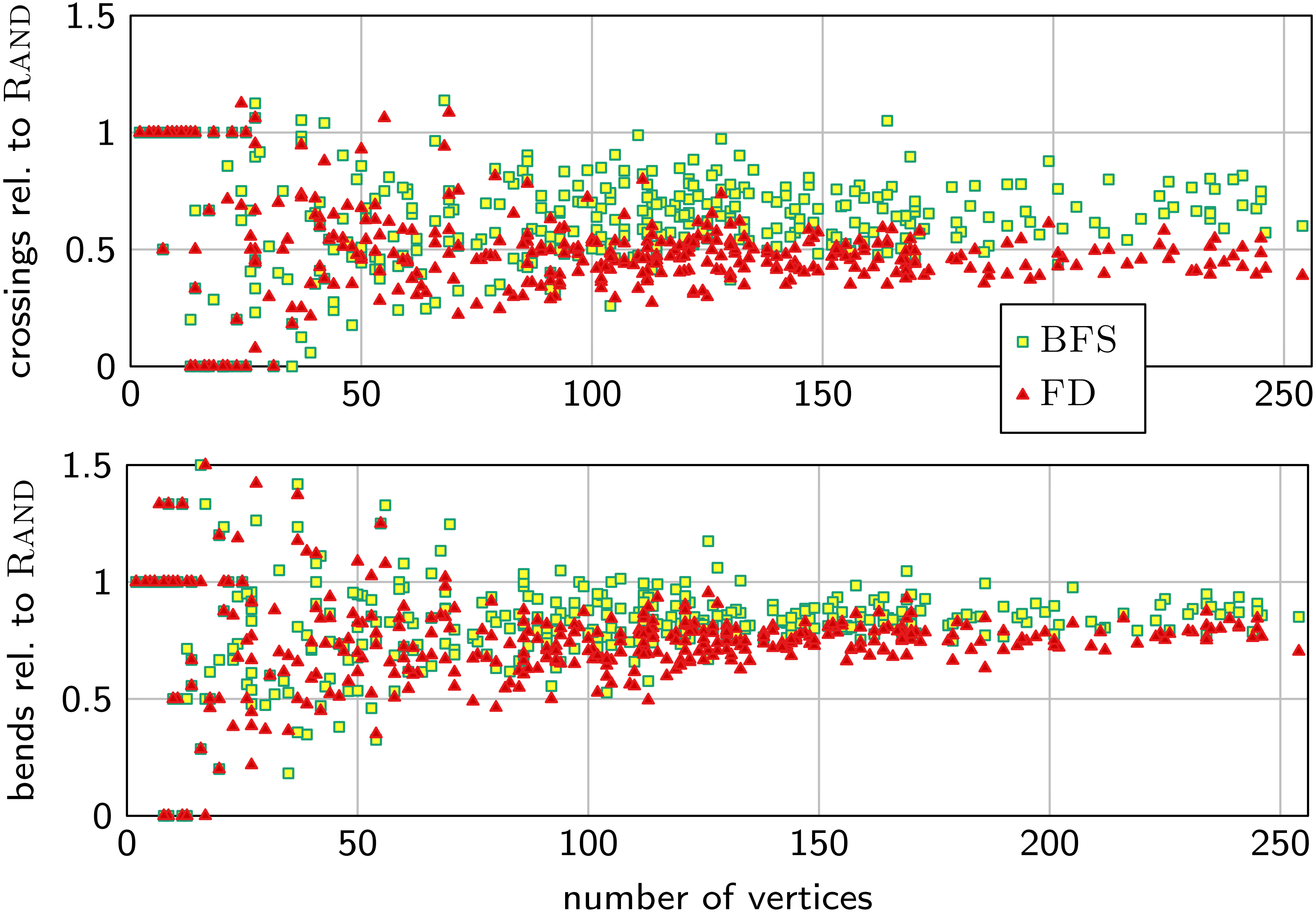
our implementation



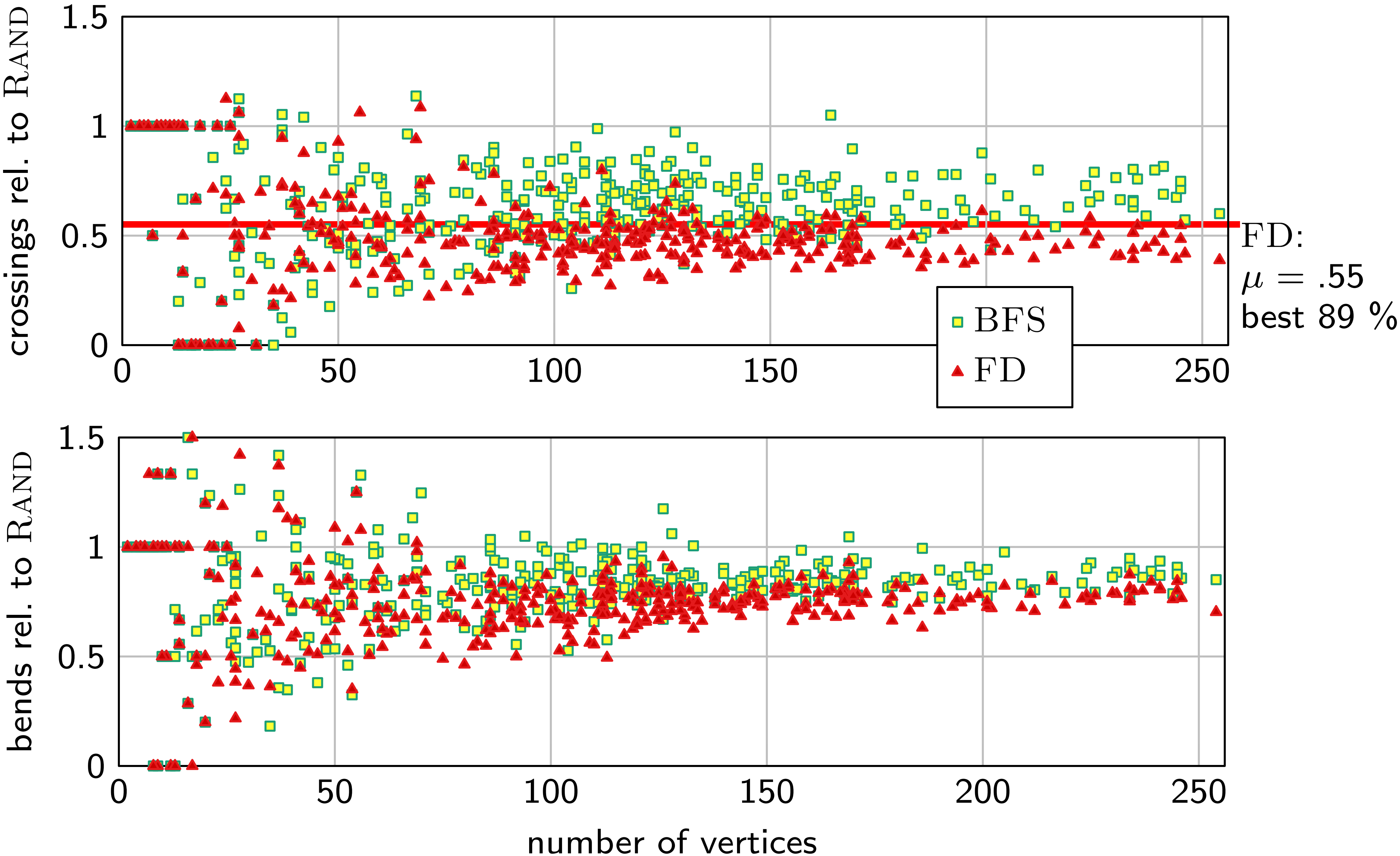


KIELER

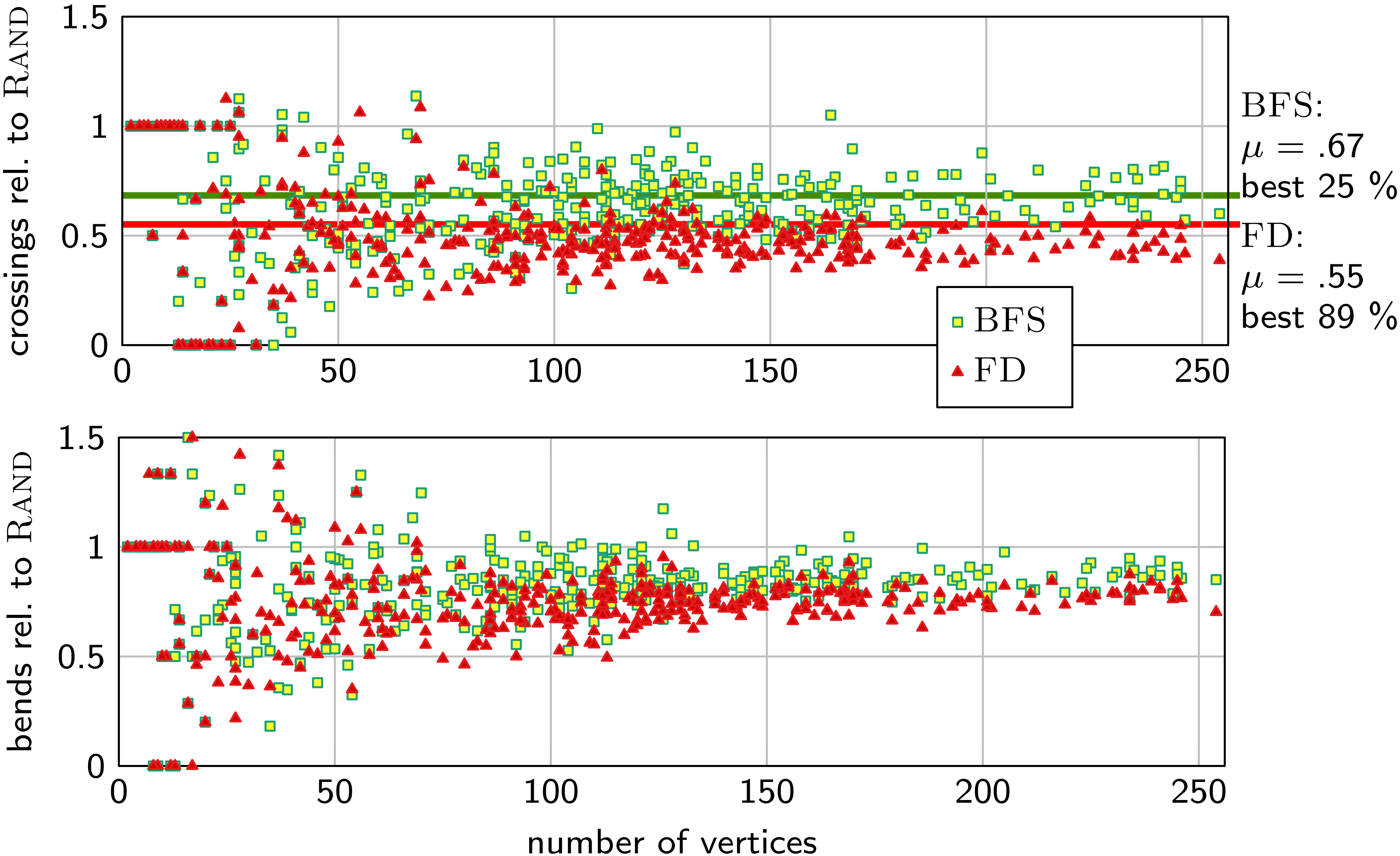
Results: Orienting Edges (Real)



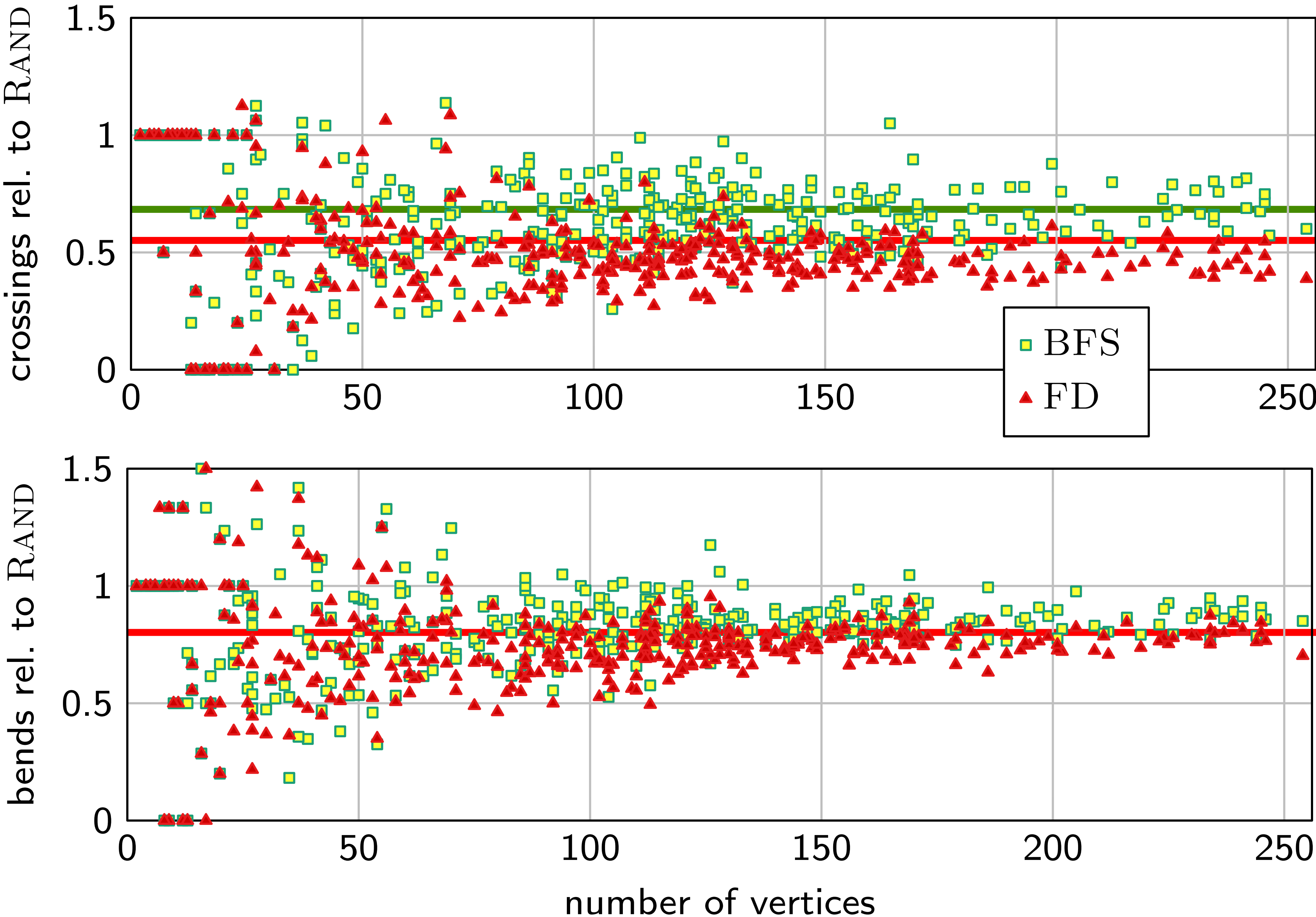
Results: Orienting Edges (Real)



Results: Orienting Edges (Real)



Results: Orienting Edges (Real)

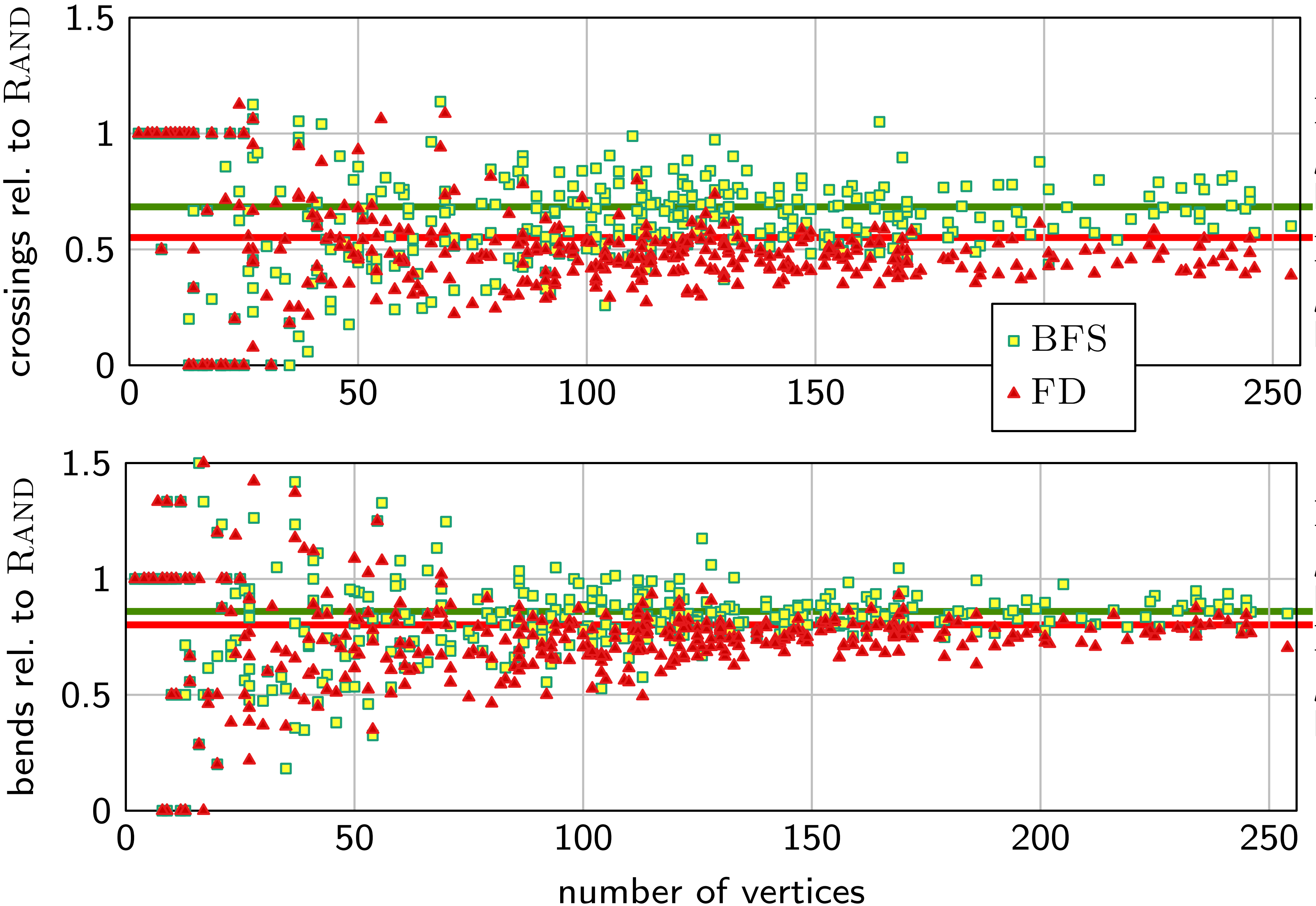


BFS:
$\mu = .67$
best 25 %

FD:
$\mu = .55$
best 89 %

FD:
$\mu = .80$
best 85 %

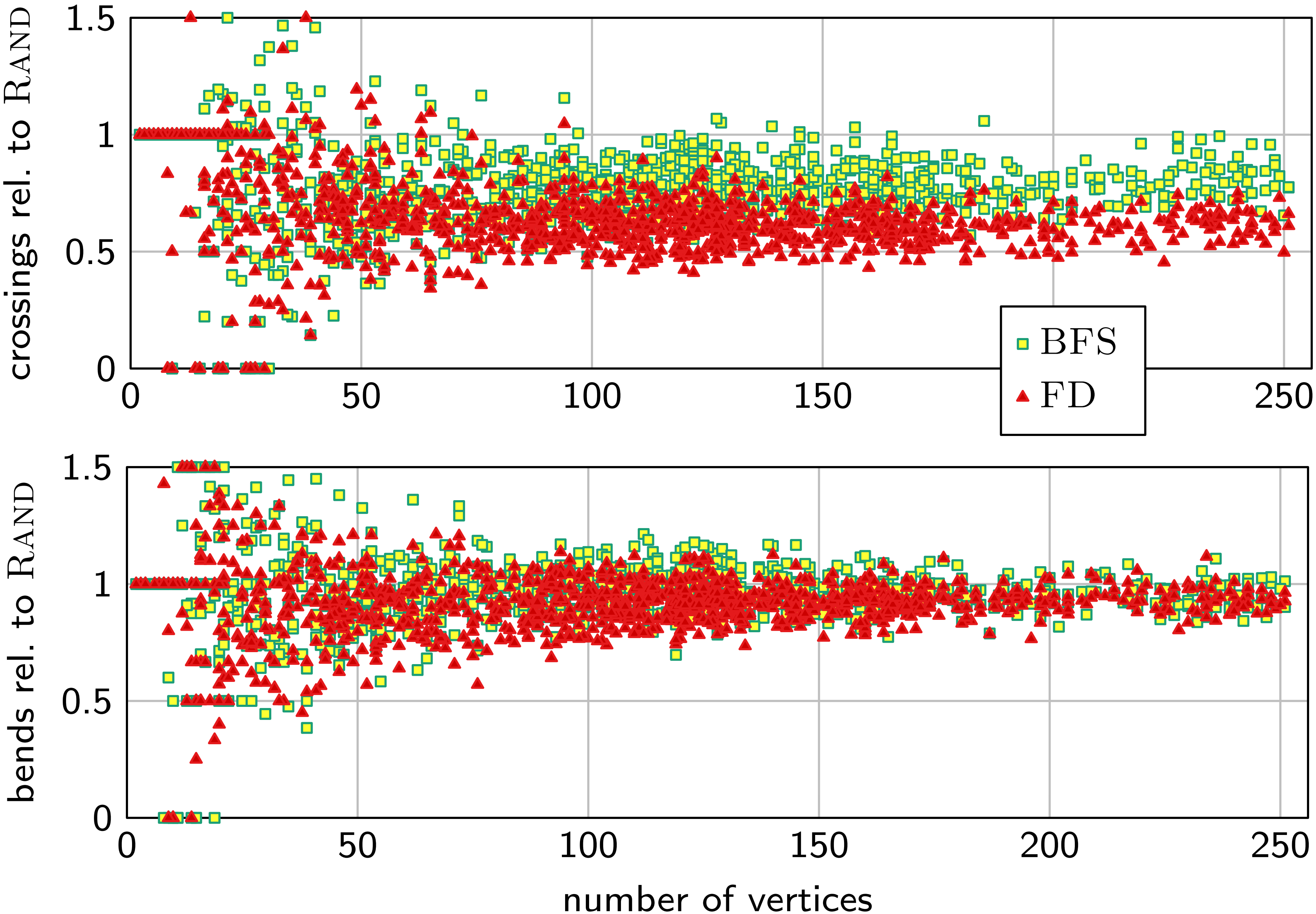
Results: Orienting Edges (Real)



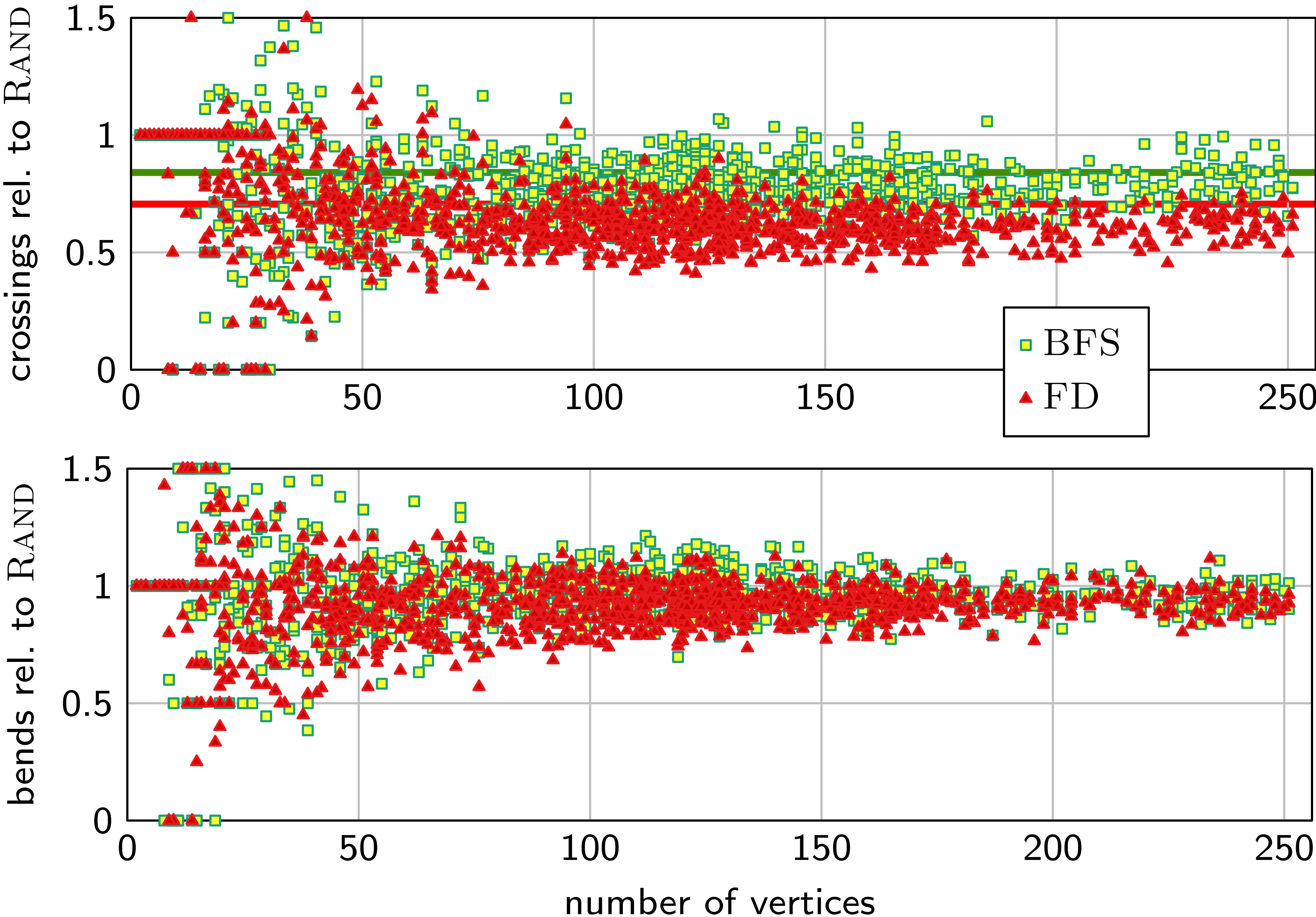
BFS:
$\mu = .67$
best 25 %
FD:
$\mu = .55$
best 89 %

BFS:
$\mu = .86$
best 20 %
FD:
$\mu = .80$
best 85 %

Results: Orienting Edges (PSEUDO)



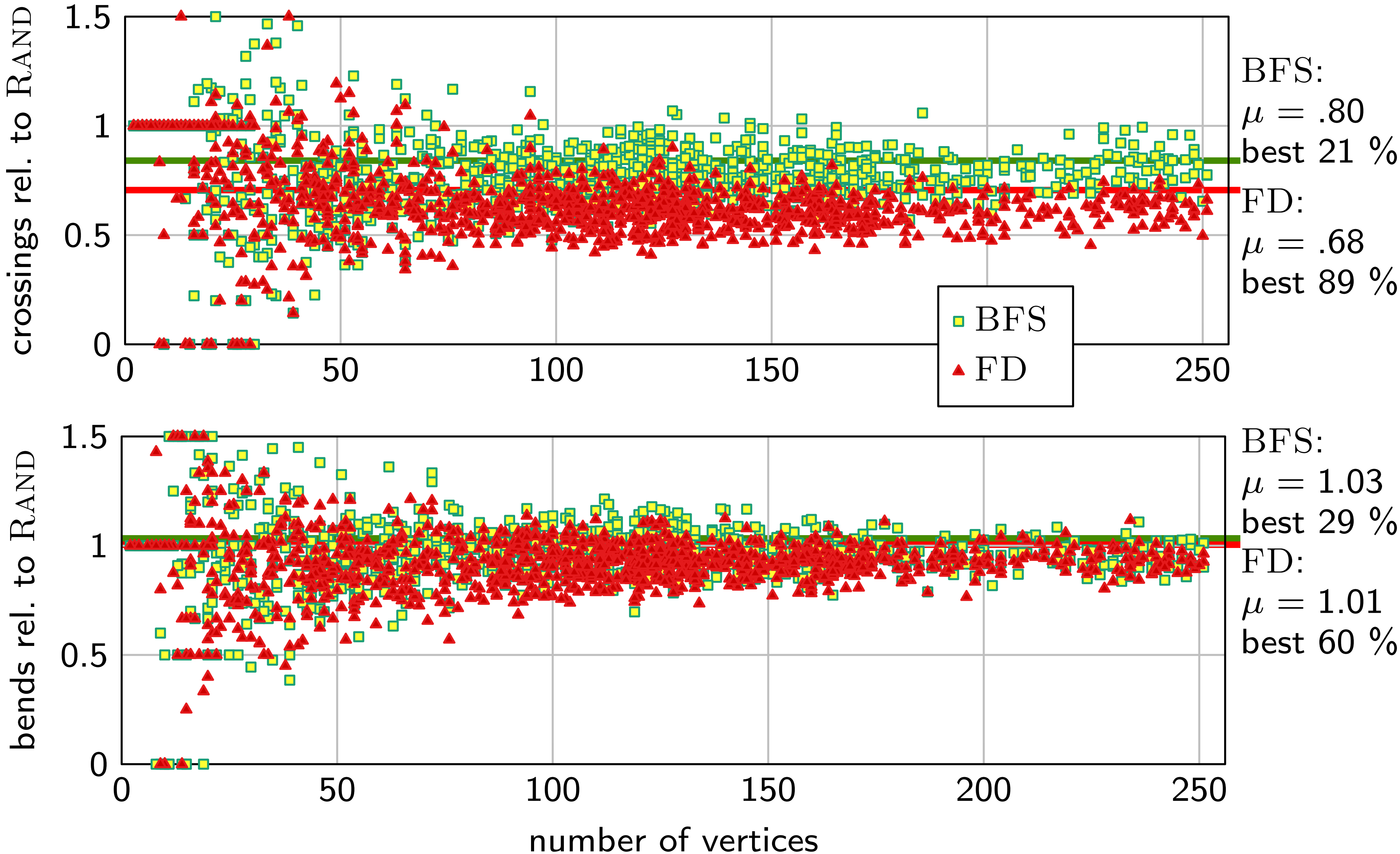
Results: Orienting Edges (PSEUDO)



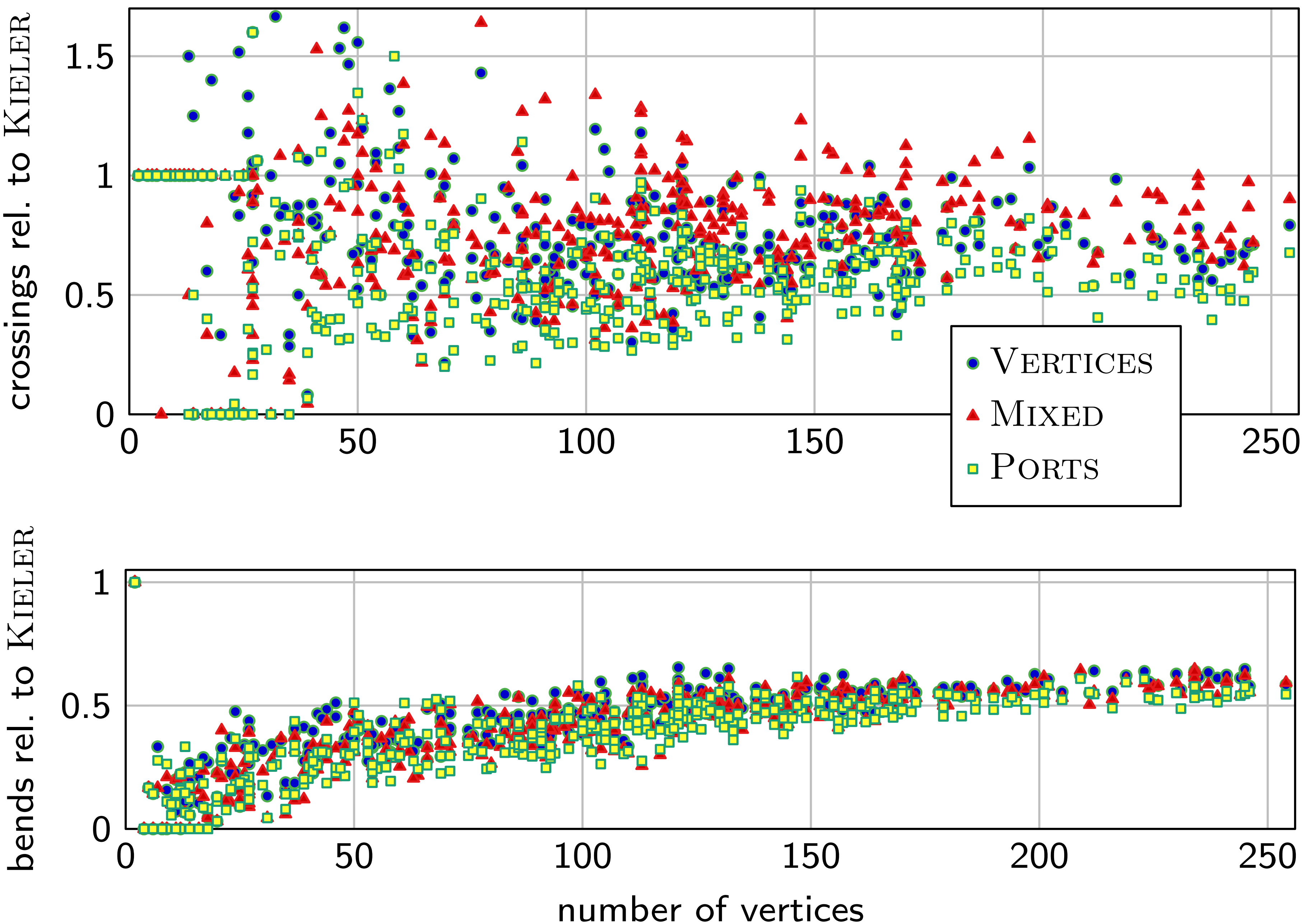
BFS:
$\mu = .80$
best 21 %

FD:
$\mu = .68$
best 89 %

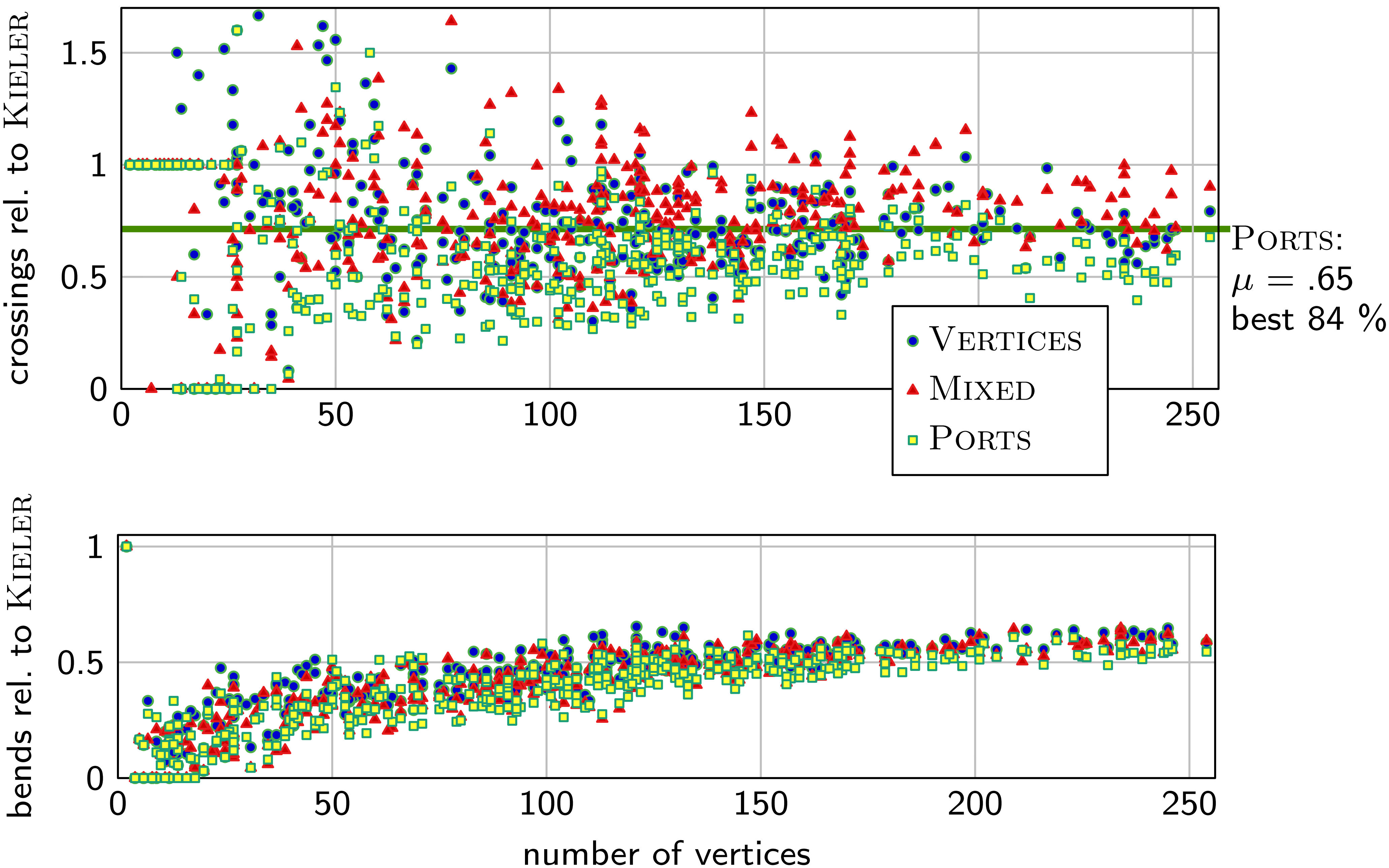
Results: Orienting Edges (PSEUDO)



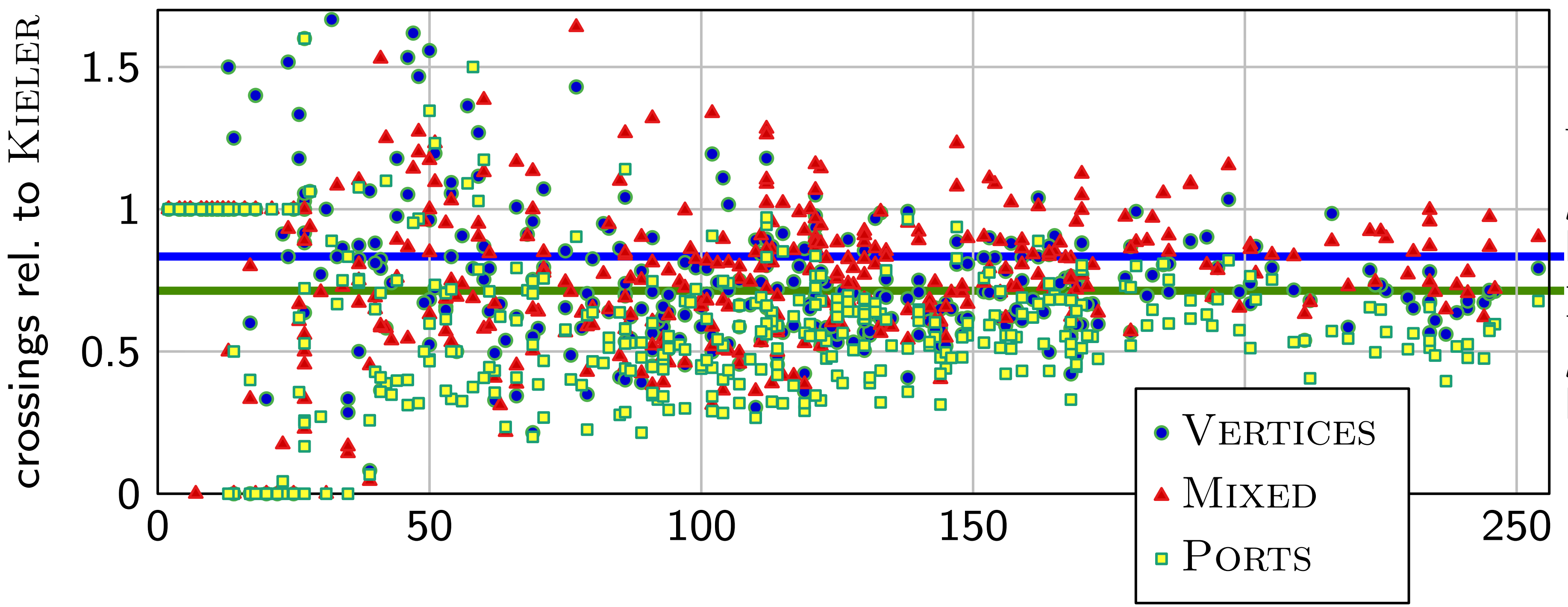
Results: Crossing Minimization (Real)



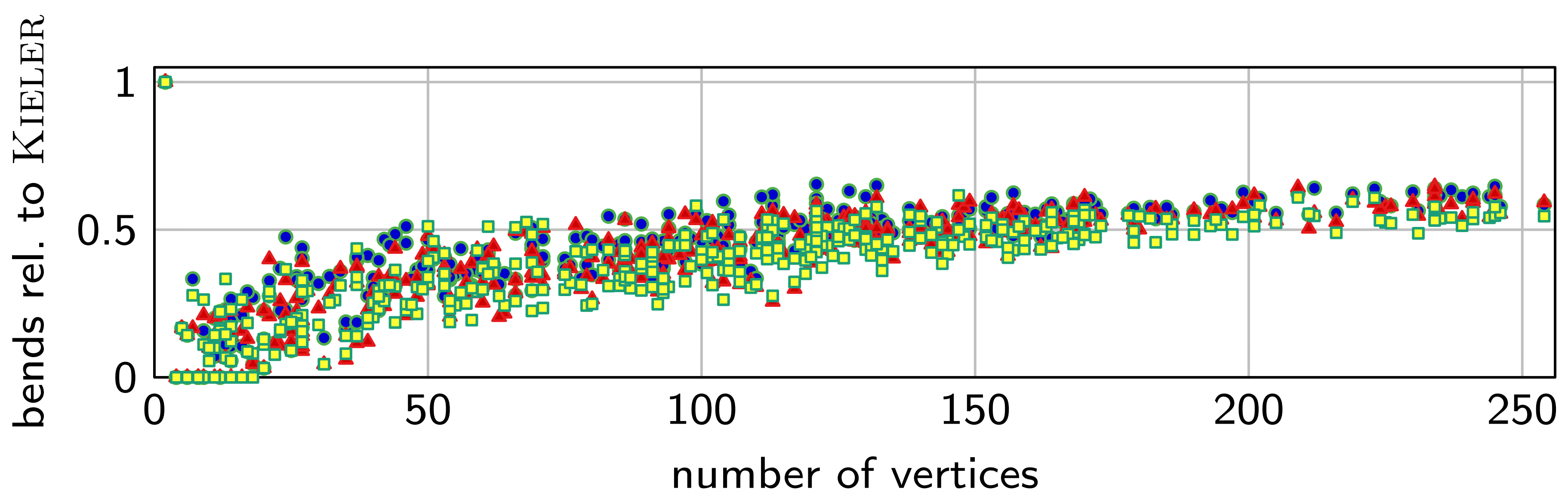
Results: Crossing Minimization (Real)



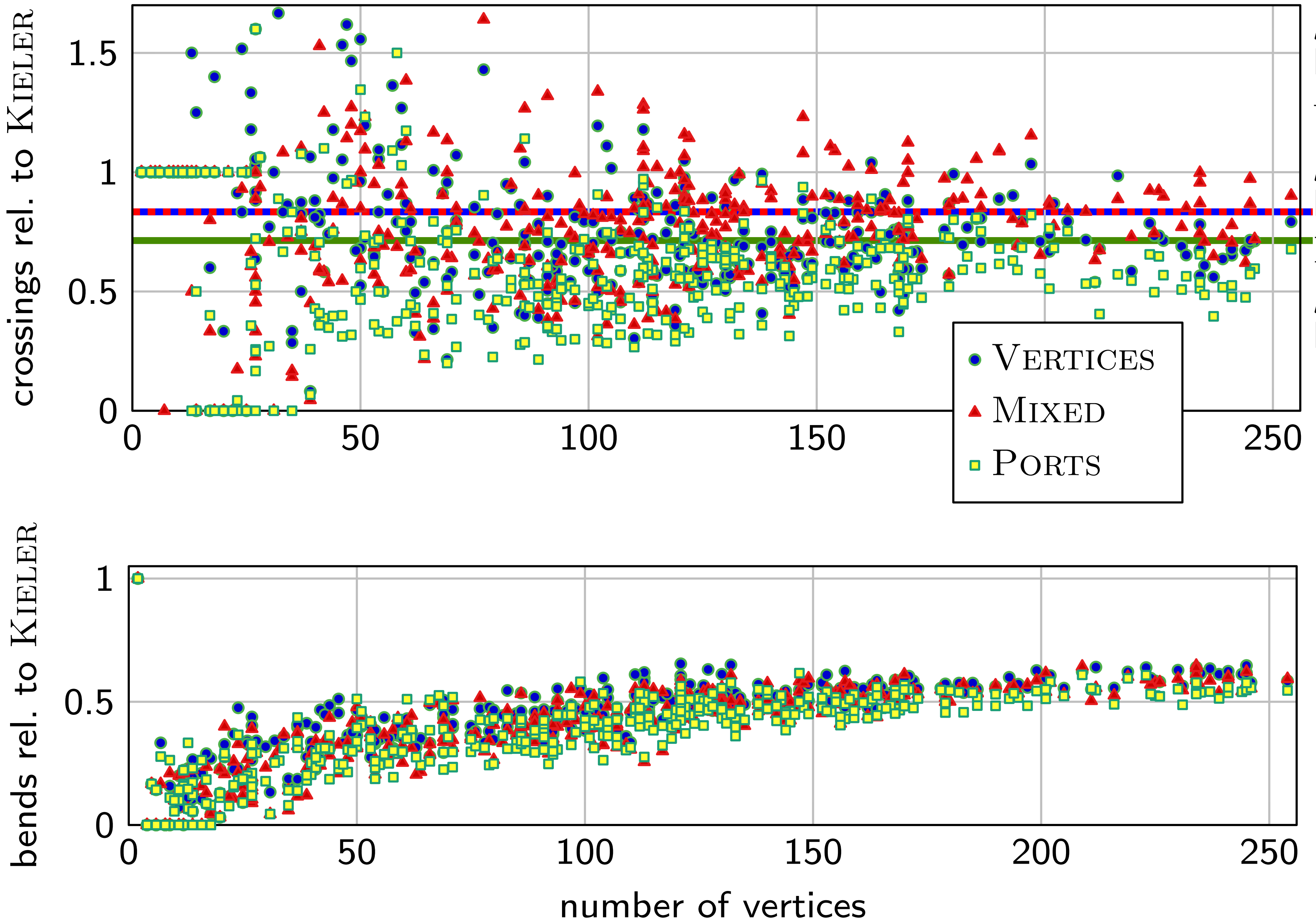
Results: Crossing Minimization (Real)



Vertices:
$\mu = .83$
best 19 %
Ports:
$\mu = .65$
best 84 %



Results: Crossing Minimization (Real)

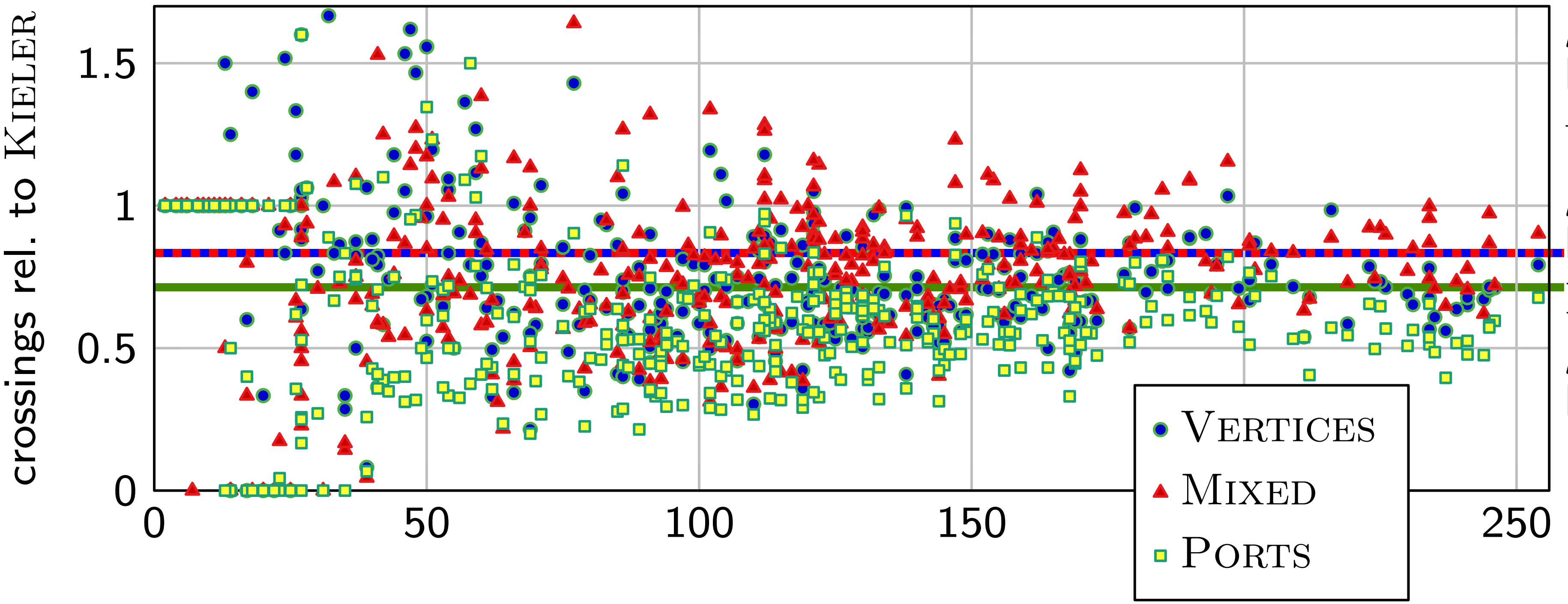


Mixed:
$\mu = .83$
best 16 %

Vertices:
$\mu = .83$
best 19 %

Ports:
$\mu = .65$
best 84 %

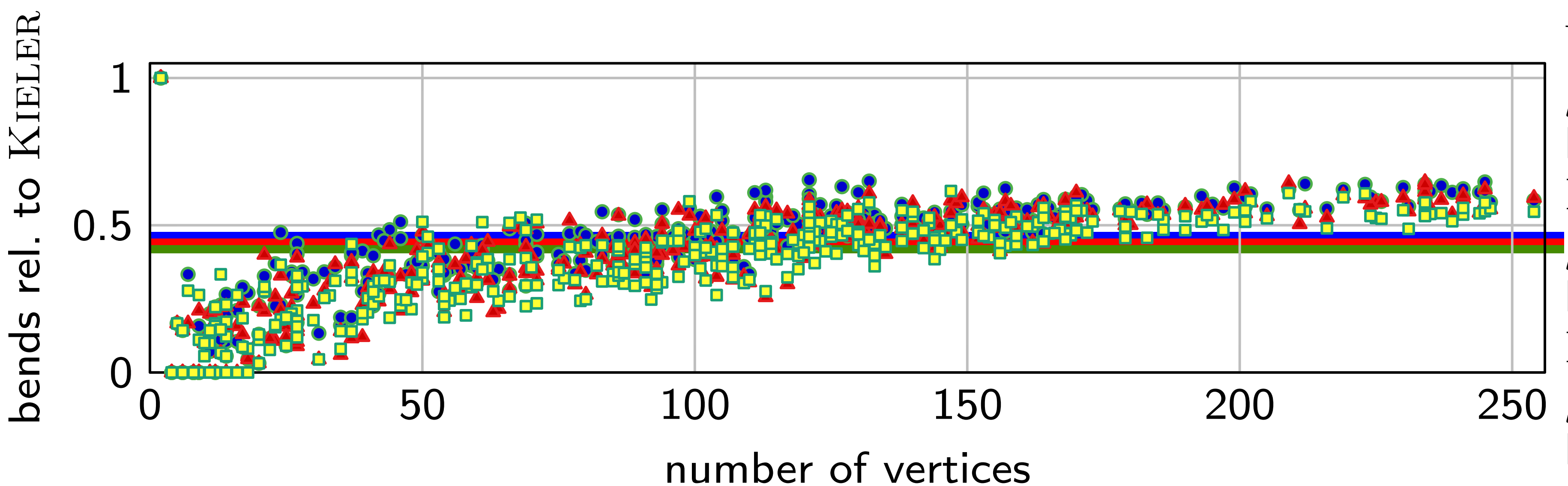
Results: Crossing Minimization (Real)



Mixed:
$\mu = .83$
best 16 %

Vertices:
$\mu = .83$
best 19 %

Ports:
$\mu = .65$
best 84 %

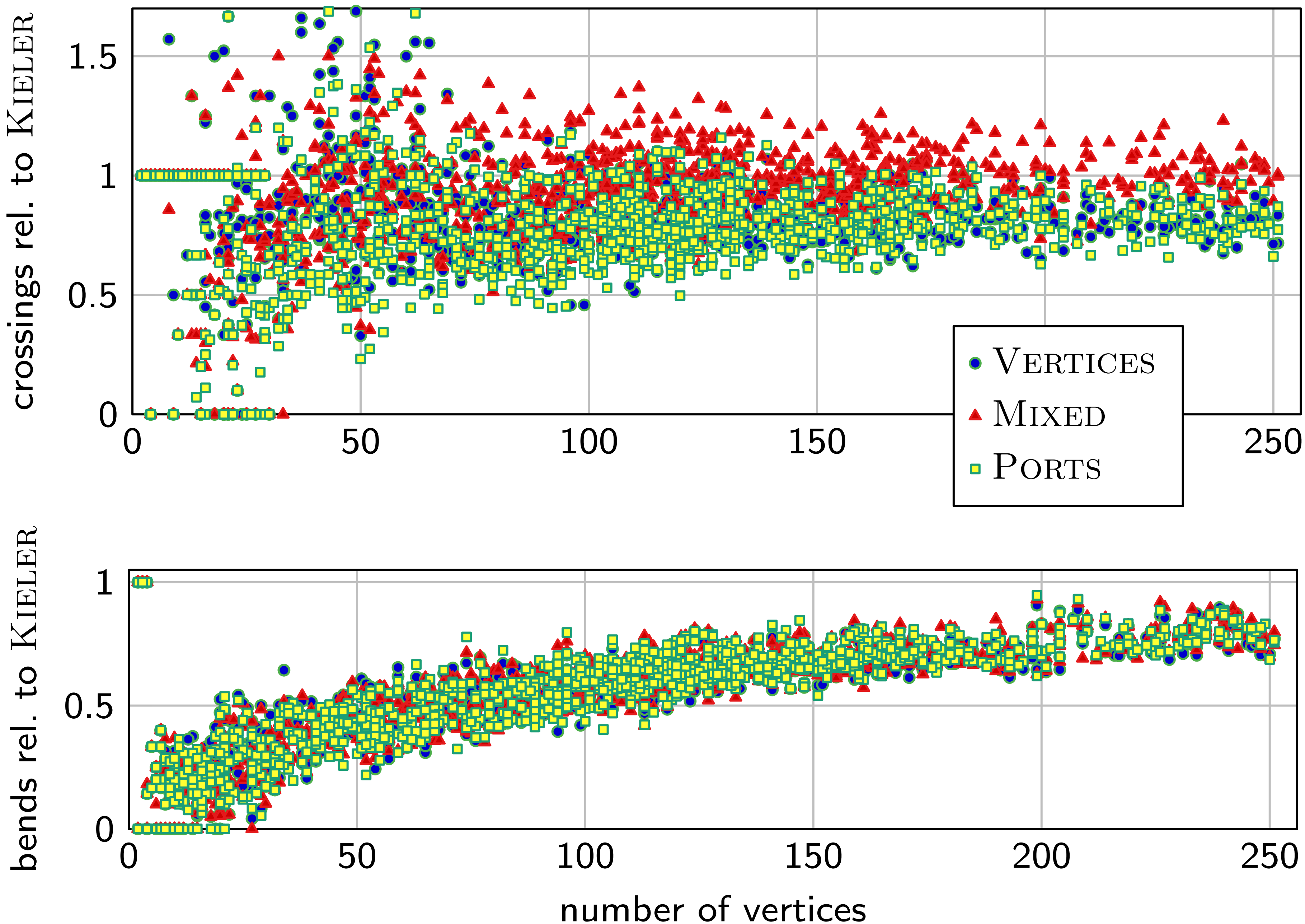


Vertices:
$\mu = .46$
best 13 %

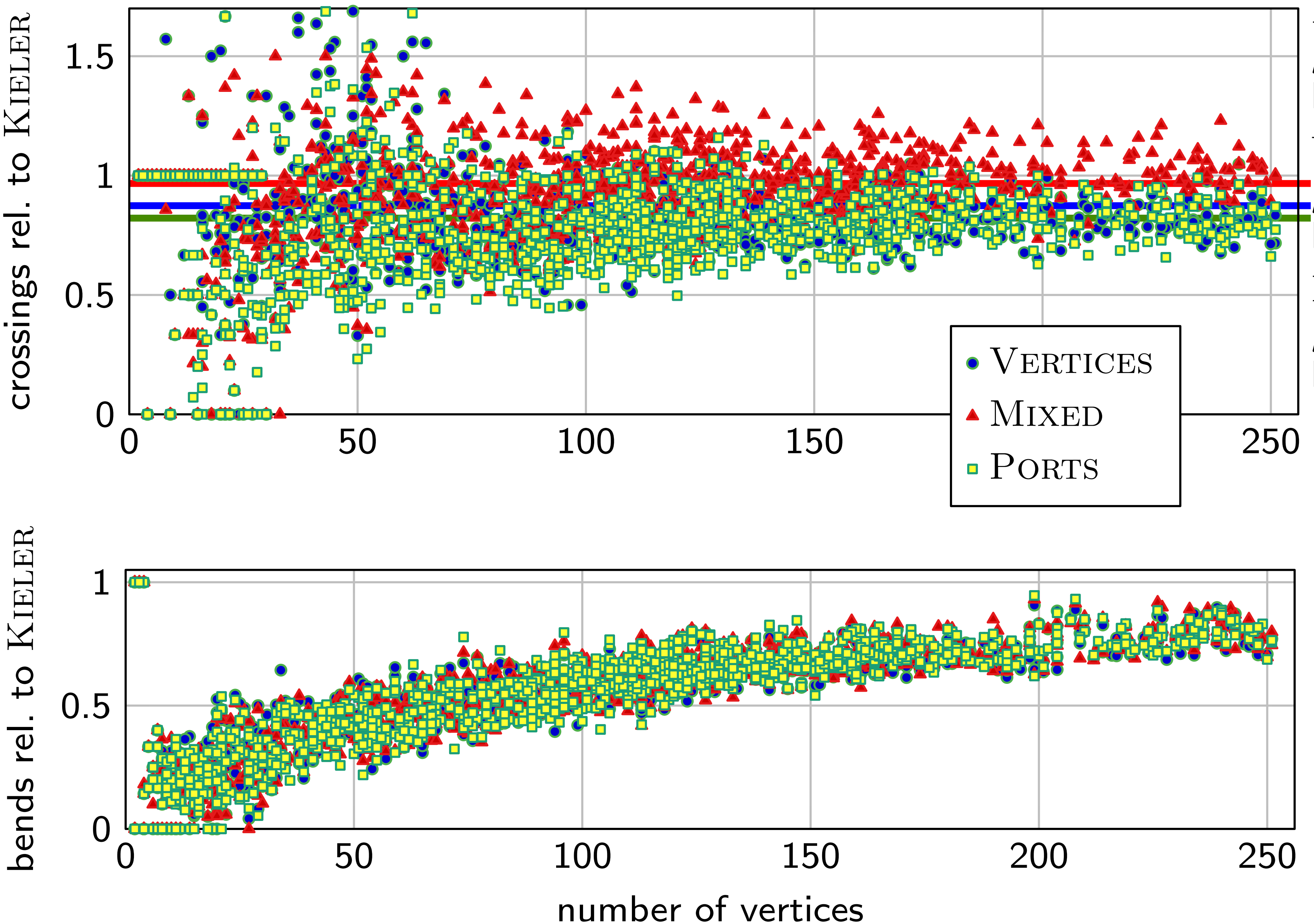
Mixed:
$\mu = .44$
best 29 %

Ports:
$\mu = .42$
best 72 %

Results: Crossing Minimization (Pseudo)



Results: Crossing Minimization (Pseudo)

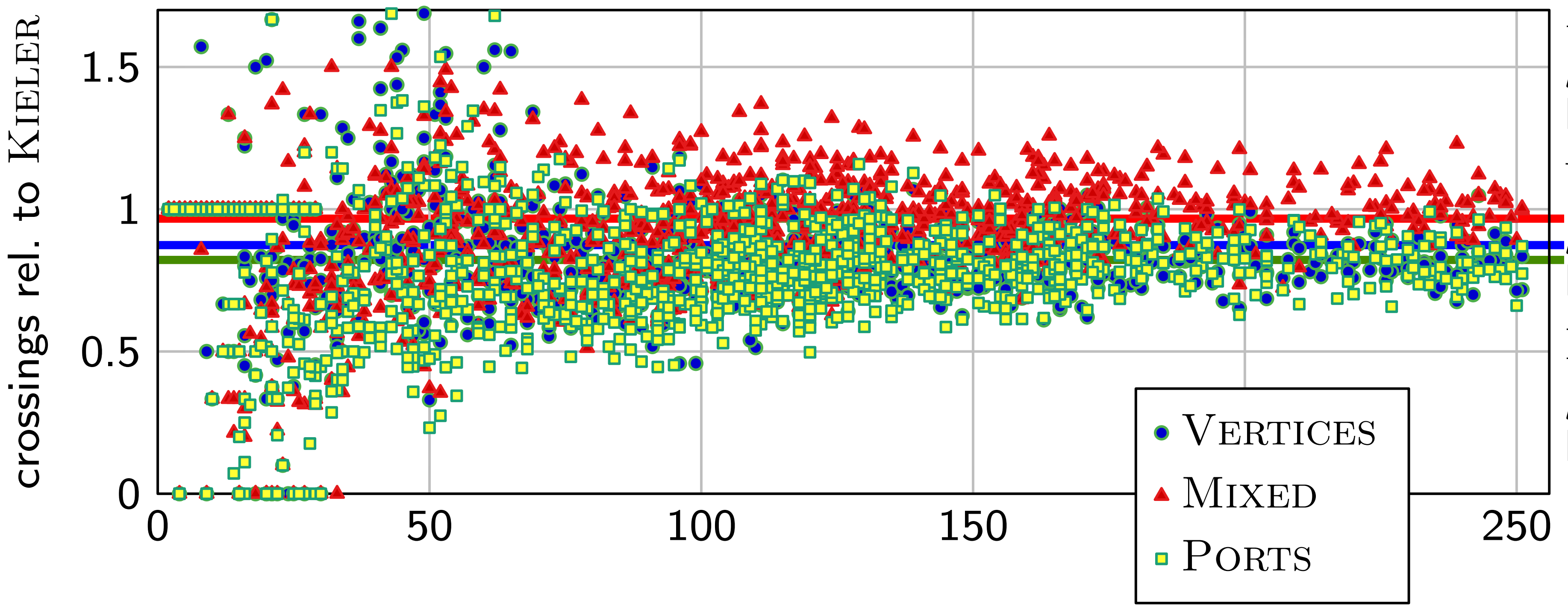


Mixed:
$\mu = .96$
best 15 %

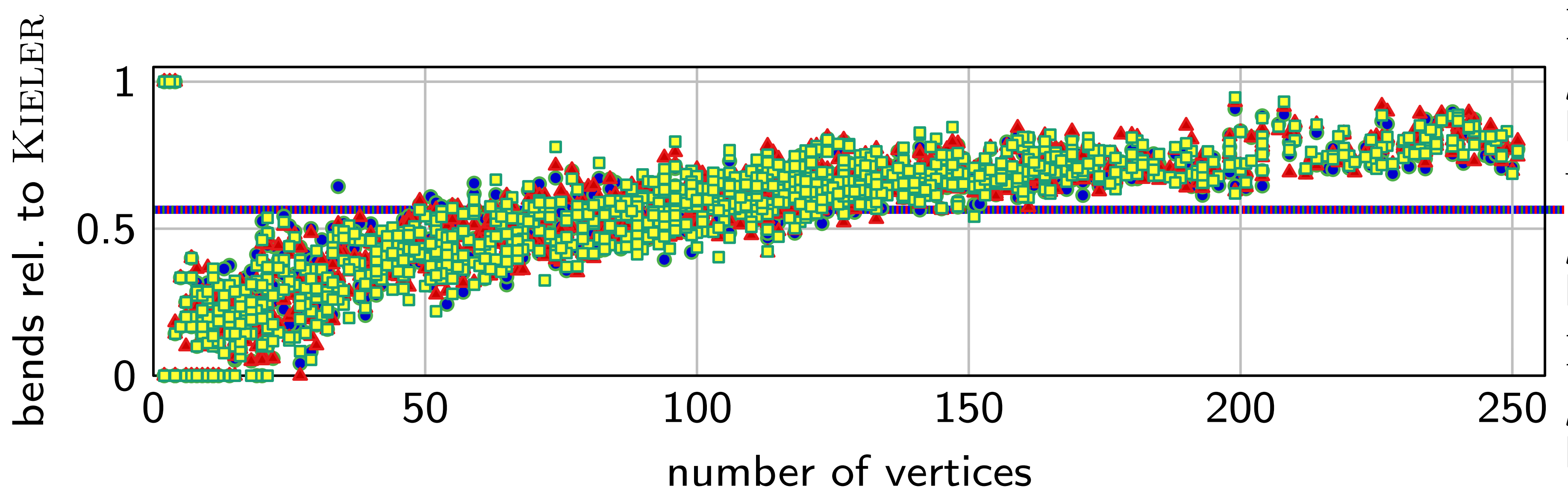
Vertices:
$\mu = .87$
best 39 %

Ports:
$\mu = .82$
best 62 %

Results: Crossing Minimization (Pseudo)

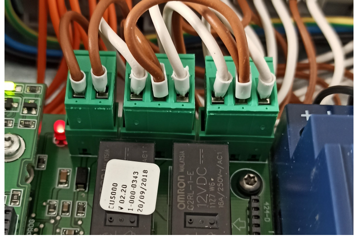


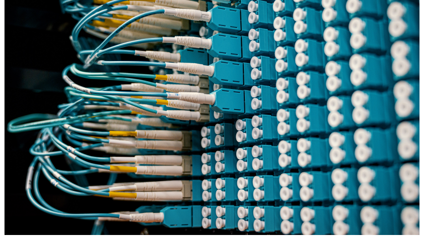
Mixed:
$\mu = .96$
best 15 %
Vertices:
$\mu = .87$
best 39 %
Ports:
$\mu = .82$
best 62 %



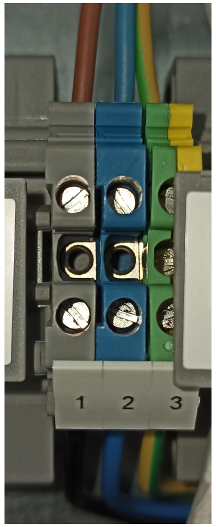
Mixed:
$\mu = .56$
best 34 %
Vertices:
$\mu = .56$
best 40 %
Ports:
$\mu = .56$
best 41 %

Conclusions

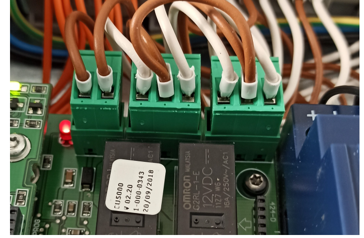


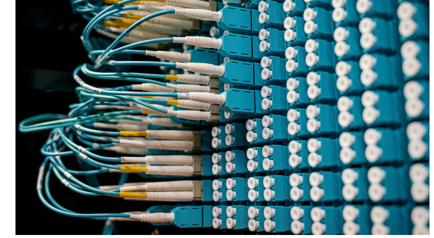


- We have extended the well-known Sugiyama framework to draw technical plans (like cable plans) that are undirected, have ports contained in (nested) port groups and plugs.



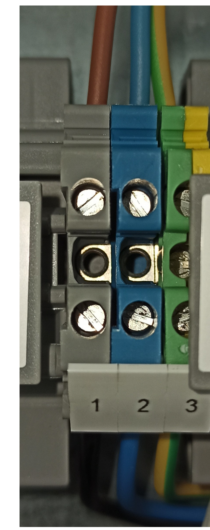
Conclusions



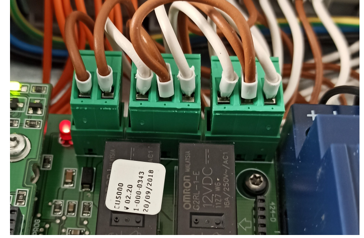


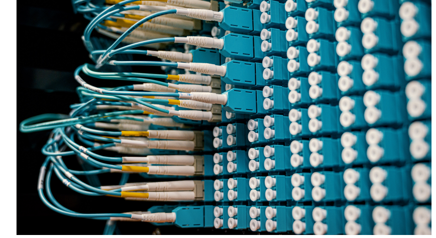
- We have extended the well-known Sugiyama framework to draw technical plans (like cable plans) that are undirected, have ports contained in (nested) port groups and plugs.

- We have implemented and experimentally evaluated our algorithm on real and on synthetic data.

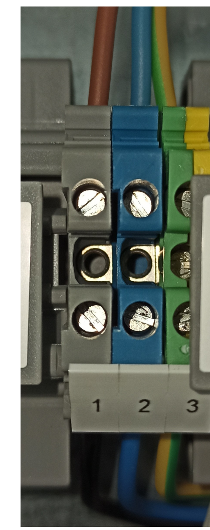


Conclusions





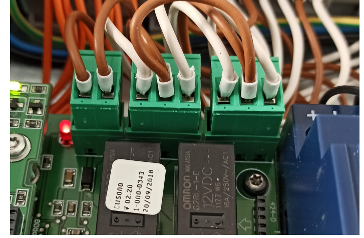
- We have extended the well-known Sugiyama framework to draw technical plans (like cable plans) that are undirected, have ports contained in (nested) port groups and plugs.

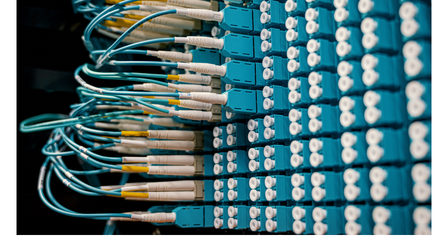


- We have implemented and experimentally evaluated our algorithm on real and on synthetic data.

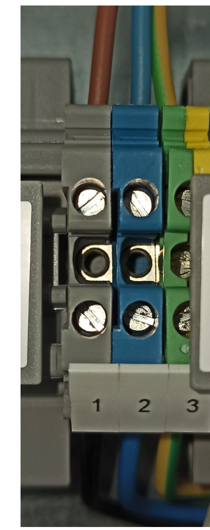
- FD was best for orienting undirected edges; PORTS was best for reducing crossings.

Conclusions





- We have extended the well-known Sugiyama framework to draw technical plans (like cable plans) that are undirected, have ports contained in (nested) port groups and plugs.

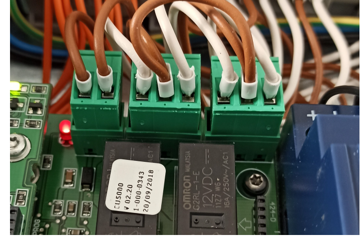


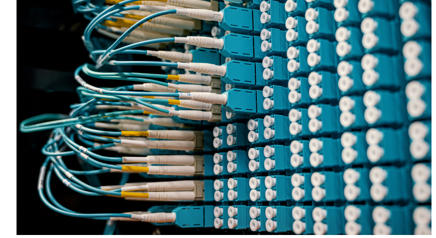
- We have implemented and experimentally evaluated our algorithm on real and on synthetic data.

- FD was best for orienting undirected edges;
  PORTS was best for reducing crossings.

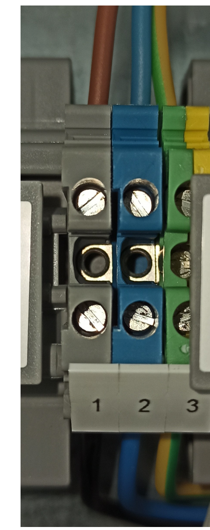
- Our variants compare well with existing implementation (KIELER) in terms of #crossings and #bends (but slower).

Conclusions





- We have extended the well-known Sugiyama framework to draw technical plans (like cable plans) that are undirected, have ports contained in (nested) port groups and plugs.



- We have implemented and experimentally evaluated our algorithm on real and on synthetic data.
- FD was best for orienting undirected edges; PORTS was best for reducing crossings.
- Our variants compare well with existing implementation (KIELER) in terms of #crossings and #bends (but slower).
- We intend to integrate our algorithm into the software of our industrial partner to see whether this statistical improvement yields advantages in practice.