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.

⇒ 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.

⇒ 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.

⇒ draw plans automatically s.t. they resemble hand-drawn plans
– orthogonal style
– vertices arranged on few layers

⇒ 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, \ldots
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, ...

Consists of 5 phases:
Consists of 5 phases:

1. cycle elimination

Consists of 5 phases:

1. cycle elimination

Consists of 5 phases:

1. cycle elimination

Consists of 5 phases:

1. cycle elimination

2. layer assignment

Consists of 5 phases:

1. cycle elimination
2. layer assignment

Consists of 5 phases:

1. cycle elimination

2. layer assignment

Consists of 5 phases:

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

Consists of 5 phases:

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

Consists of 5 phases:

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

Consists of 5 phases:

1. cycle elimination
2. layer assignment
3. crossing minimization

Consists of 5 phases:

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

Consists of 5 phases:

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

Consists of 5 phases:

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

Consists of 5 phases:

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

Consists of 5 phases:

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

contains NP-hard tasks

Consists of 5 phases:

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

contains NP-hard tasks

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 $E$
- 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 \( E \)
- 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 $E \ni e : \{p_1, p_2\} \in \binom{P}{2}$
- 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 $E \ni e : \{p_1, p_2\} \in \binom{P}{2}$
- 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

Orthogonal drawing style:
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 $\geq h(v)$
Definitions

Extension of a graph to a *port graph* $G = (V, P, E)$:
- vertex set $V$
- edge set $E \ni e: \{p_1, p_2\} \in (P^2)$
- 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 $E \ni e : \{p_1, p_2\} \in \binom{P}{2}$
- 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

• Spönnemann 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):
Spönnemann 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

- Spönnemann 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

- Spönnemann 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

- Spönenmann 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 implementation 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 \texttt{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 $\text{Free}, \text{FixedSide}, \text{FixedOrder}$)
  - can be assigned to a vertex side or $\text{Free}$
Contribution

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

- Port groups (generalizing port constraints $\text{Free, FixedSide, FixedOrder}$)
  - can be assigned to a vertex side or $\text{Free}$
  - 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 `\texttt{Free, FixedSide, FixedOrder}`)
  - can be assigned to a vertex side or `\texttt{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 \texttt{Free}, \texttt{FixedSide}, \texttt{FixedOrder})
  - can be assigned to a vertex side or \texttt{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 \texttt{FREE, FIXEDSIDE, FIXEDORDER})
  - can be assigned to a vertex side or \texttt{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 \texttt{Free}, \texttt{FixedSide}, \texttt{FixedOrder})
  - can be assigned to a vertex side or \texttt{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 assigned 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 assigned 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 \texttt{Free, FixedSide, FixedOrder})
  - can be assigned to a vertex side or \texttt{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 \texttt{Free}, \texttt{FixedSide}, \texttt{FixedOrder})
  - can be assigned to a vertex side or \texttt{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 $\text{Free}$, $\text{FixedSide}$, $\text{FixedOrder}$)
  - can be assigned to a vertex side or $\text{Free}$
  - 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 \texttt{Free, FixedSide, FixedOrder})
  - can be assigned to a vertex side or \texttt{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

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)
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

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:
1. **cycle elimination**

2. **layer assignment**

3. **crossing minimization**

   - Well-established barycenter heuristic with respect to:

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

Well-established barycenter heuristic with respect to:

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:

   • 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
Our Extensions to the Sugiyama Framework

1. **Cycle Elimination**

2. **Layer Assignment**

3. **Crossing Minimization**

   Well-established barycenter heuristic with respect to:
   - **Vertices** – sort ports afterwards
   - **Ports** – sort port groups & vtc.
     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**

   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
   - 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
   (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](https://github.com/j-zink-wuerzburg).../praline
  .../pseudo-praline-plan-generation
Example: (anonymized) plan from Real

Kieler

our implementation
Example: plan from PSEUDO

our implementation
Results: Orienting Edges (REAL)
Results: Orienting Edges (REAL)

FD: $\mu = 0.55$
best 89%

- Crossings rel. to RAND
- Bends rel. to RAND vs. number of vertices

- BFS
- FD
Results: Orienting Edges (REAL)

BFS:
\[ \mu = 0.67 \]
best 25%

FD:
\[ \mu = 0.55 \]
best 89%

Number of vertices:
- BFS
- FD

Crossings relative to RAND
- BFS
- FD

Bends relative to RAND
- BFS
- FD

Number of vertices vs. crossings/bends

Diagram showing scatter plots for BFS and FD, with lines indicating best performance percentages.
Results: Orienting Edges (REAL)

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

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

FD:
$$\mu = .80$$
best 85 %
Results: Orienting Edges (REAL)

- **BFS**: $\mu = 0.67$, best 25%
- **FD**: $\mu = 0.55$, best 89%

Bends rel. to RAND:
- **BFS**: $\mu = 0.86$, best 20%
- **FD**: $\mu = 0.80$, best 85%
Results: Orienting Edges (PSEUDO)

The graphs depict the relative number of crossings and bends in relation to the baseline Rand. The x-axis represents the number of vertices, ranging from 0 to 250. There are two main plots:

1. **Crossings rel. to Rand**: The top plot shows the relative number of crossings. The data points are color-coded and marked with different symbols for different algorithms.

2. **Bends rel. to Rand**: The bottom plot illustrates the relative number of bends. Similar to the crossings plot, the data points are color-coded and marked with symbols for different algorithms.

The graphs provide a visual comparison of the performance of different algorithms in terms of edge orientation efficiency.
Results: Orienting Edges (PSEUDO)

FD:
\[ \mu = 0.68 \]
best 89%

BFS:
\[ \mu = 0.80 \]
best 21%

BFS:
\[ \mu = 0.80 \]
best 21%

FD:
\[ \mu = 0.68 \]
best 89%

number of vertices

bends rel. to RAND

crossings rel. to RAND
Results: Orienting Edges (PSEUDO)

FD:
\( \mu = 0.68 \)
best 89%

BFS:
\( \mu = 0.80 \)
best 21%

FD:
\( \mu = 1.01 \)
best 60%

BFS:
\( \mu = 1.03 \)
best 29%
Results: Crossing Minimization (REAL)
Results: Crossing Minimization (REAL)

Ports: $\mu = .65$
best 84 %
Results: Crossing Minimization (REAL)

Vertices:
- $\mu = 0.83$
- Best 19%

Ports:
- $\mu = 0.65$
- Best 84%

Graphs showing crossings and bends relative to KIELER.
Results: Crossing Minimization (REAL)

**Mixed:**
\[ \mu = 0.83 \]
best 16%

**Vertices:**
\[ \mu = 0.83 \]
best 19%

**Ports:**
\[ \mu = 0.65 \]
best 84%

---

![Graph showing crossings and bends relative to KIELER for different categories: Vertices, Mixed, and Ports. The graphs display the number of vertices on the x-axis and crossings or bends on the y-axis.](Image)
Results: Crossing Minimization (REAL)

Mixed:
μ = .83
best 16%

Vertices:
μ = .83
best 19%

Ports:
μ = .65
best 84%

Mixed:
μ = .44
best 29%

Vertices:
μ = .46
best 13%

Ports:
μ = .42
best 72%
Results: Crossing Minimization (PSEUDO)

![Diagram showing crossings and bends relative to Kieler]

- **Crossings rel. to Kieler**
  - Vertical axis: Number of vertices
  - Horizontal axis: Number of vertices
  - Data points for **VERTICES**, **Mixed**, and **PORTS**

- **Bends rel. to Kieler**
  - Vertical axis: Number of vertices
  - Horizontal axis: Number of vertices
  - Data points for **VERTICES**, **Mixed**, and **PORTS**
Results: Crossing Minimization (PSEUDO)

**Mixed:**
\[ \mu = 0.96 \]
best 15%

**Vertices:**
\[ \mu = 0.87 \]
best 39%

**Ports:**
\[ \mu = 0.82 \]
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.