Algorithms for Graph Visualization

Summer Semester 2017
Lecture #7

Hierarchical Drawings

References:
Drawing Graphs: Methods and Models (Ch. 5)
Graph Drawing: Algorithms for the Visualization of Graphs (Ch. 9)
(based on slides from Marcus Krug, KIT)
Example
Example
Example
Example

E-Mail-Graph between groups in Computer Science, KIT
Hierarchical Drawing

Problem statement:

- **Input:** directed graph $D = (V, A)$
- **Output:** Drawing of $D$ which *closely* reproduces the hierarchical properties of $D$. 
Hierarchical Drawing

Problem statement:

- Input: directed graph $D = (V, A)$
- Output: Drawing of $D$ which *closely* reproduces the hierarchical properties of $D$.

Desireable Properties

- vertices occur on (few) horizontal lines
- edges directed upwards
- edge crossings minimized
- edges as vertical, straight, and short as possible
- vertices evenly spaced
Hierarchical Drawing

Problem statement:

- Input: directed graph \( D = (V, A) \)
- Output: Drawing of \( D \) which *closely* reproduces the hierarchical properties of \( D \).

Desireable Properties

- vertices occur on (few) horizontal lines
- edges directed upwards
- edge crossings minimized
- edges as vertical, straight, and short as possible
- vertices evenly spaced
Hierarchical Drawing

Problem statement:

- **Input:** directed graph $D = (V, A)$
- **Output:** Drawing of $D$ which *closely* reproduces the hierarchical properties of $D$.

Desireable Properties

- vertices occur on (few) horizontal lines
- edges directed upwards
- edge crossings minimized
- edges as vertical, straight, and short as possible
- vertices evenly spaced
Hierarchical Drawing

Problem statement:

• Input: directed graph $D = (V, A)$
• Output: Drawing of $D$ which *closely* reproduces the hierarchical properties of $D$.

Desireable Properties

• vertices occur on (few) horizontal lines
• edges directed upwards
• edge crossings minimized
Hierarchical Drawing

Problem statement:

- **Input:** directed graph \( D = (V, A) \)
- **Output:** Drawing of \( D \) which *closely* reproduces the hierarchical properties of \( D \).

Desirable Properties

- vertices occur on (few) horizontal lines
- edges directed upwards
- edge crossings minimized
- edges as vertical, straight, and short as possible
Hierarchical Drawing

Problem statement:

- **Input:** directed graph $D = (V, A)$
- **Output:** Drawing of $D$ which *closely* reproduces the hierarchical properties of $D$.

Desirable Properties

- vertices occur on (few) horizontal lines
- edges directed upwards
- edge crossings minimized
- edges as vertical, straight, and short as possible
- vertices evenly spaced
Hierarchical Drawing

Problem statement:

- **Input:** directed graph \( D = (V, A) \)
- **Output:** Drawing of \( D \) which *closely* reproduces the hierarchical properties of \( D \).

Desireable Properties

- vertices occur on (few) horizontal lines
- edges directed upwards
- edge crossings minimized
- edges as vertical, straight, and short as possible
- vertices evenly spaced

Criteria can be contradictory!
Classical Approach

[Sugiyama, Tagawa, Toda ’81]

Input

```
1 → 6 → 7
```

```
4 ← 2 ← 5
```

```
3
```

```
```
Classical Approach [Sugiyama, Tagawa, Toda ’81]

Input $\rightarrow$ Cycle breaking

Diagram:

1. Input graph (left)
2. Cycle breaking graph (right)
Classical Approach

[Sugiyama, Tagawa, Toda ’81]
Classical Approach

[Sugiyama, Tagawa, Toda ’81]

Input → Cycle breaking → Leveling

Crossing minimization
Classical Approach

[Sugiyama, Tagawa, Toda ’81]
Classical Approach

[Sugiyama, Tagawa, Toda '81]

Input \rightarrow Cycle breaking \rightarrow Leveling

Cycle breaking

Leveling

Input

Cycle breaking

Leveling

Input

Cycle breaking

Leveling

Input

Cycle breaking

Leveling

Input

Classical Approach

[Sugiyama, Tagawa, Toda '81]
Step 1: Cycle Breaking

Approach

- Find minimum set $A^*$ of edges which are not upwards.
- Remove $A^*$ and insert reversed edges.
Step 1: Cycle Breaking

Approach

- Find minimum set $A^*$ of edges which are not upwards.
- Remove $A^*$ and insert reversed edges.

Problem Minimum Feedback Arc Set (FAS):

- Input: directed graph $D = (V, A)$
- Output: min. set $A^* \subseteq A$, so that $D - A^*$ acyclic
Step 1: Cycle Breaking

Approach

- Find minimum set $A^*$ of edges which are not upwards.
- Remove $A^*$ and insert reversed edges.

Problem Minimum Feedback Arc Set (FAS):

- Input: directed graph $D = (V, A)$
- Output: min. set $A^* \subseteq A$, so that $D - A^*$ acyclic

... NP-hard :-(

\[
\begin{array}{cccc}
1 & 6 & 7 & 2
\end{array}
\]
Step 1: Cycle Breaking

Approach

- Find minimum set $A^*$ of edges which are not upwards.
- Remove $A^*$ and insert reversed edges.

Problem Minimum Feedback Arc Set (FAS):

- Input: directed graph $D = (V, A)$
- Output: min. set $A^* \subseteq A$, so that $D - A^* + A_r^*$ acyclic

... NP-hard :-(
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph $D = (V, A)$)

$A' \leftarrow \emptyset$ (these will be the edges we keep)

return $(V, A')$
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph $D = (V, A)$)

$A' \leftarrow \emptyset$ (these will be the edges we keep)

\begin{verbatim}
foreach $v \in V$ do
  if outdeg($v$) > indeg($v$) then
    $A' \leftarrow A' \cup \text{out}(v)$
  else
    $A' \leftarrow A' \cup \text{in}(v)$
  $A \leftarrow A \setminus (\text{out}(v) \cup \text{in}(v))$
\end{verbatim}

return $(V, A')$
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph $D = (V, A)$)

$A' \leftarrow \emptyset$ (these will be the edges we keep)

**foreach** $v \in V$ **do**

- **if** $\text{outdeg}(v) > \text{indeg}(v)$ **then**
  - $A' \leftarrow A' \cup \text{out}(v)$
- **else**
  - $A' \leftarrow A' \cup \text{in}(v)$

$A \leftarrow A \setminus (\text{out}(v) \cup \text{in}(v))$

**return** $(V, A')$
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph $D = (V, A)$)

$A' \leftarrow \emptyset$ (these will be the edges we keep)

foreach $v \in V$ do

if $\text{outdeg}(v) > \text{indeg}(v)$ then

$A' \leftarrow A' \cup \text{out}(v)$

else

$A' \leftarrow A' \cup \text{in}(v)$

$A \leftarrow A \backslash (\text{out}(v) \cup \text{in}(v))$

return $(V, A')$

• Timing: ?
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph $D = (V, A)$)

$A' \leftarrow \emptyset$ (these will be the edges we keep)

foreach $v \in V$ do

  if $\text{outdeg}(v) > \text{indeg}(v)$ then
    $A' \leftarrow A' \cup \text{out}(v)$
  else
    $A' \leftarrow A' \cup \text{in}(v)$

$A \leftarrow A \setminus (\text{out}(v) \cup \text{in}(v))$

return $(V, A')$

- Timing: $O(V + A)$
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph \( D = (V, A) \))

\[ A' \leftarrow \emptyset \text{ (these will be the edges we keep)} \]

\begin{align*}
\text{foreach } v \in V \text{ do} & \\
\quad \text{if } \text{outdeg}(v) > \text{indeg}(v) \text{ then} & \\
\qquad A' & \leftarrow A' \cup \text{out}(v) \\
\quad \text{else} & \\
\qquad A' & \leftarrow A' \cup \text{in}(v) \\
\qquad A & \leftarrow A \setminus (\text{out}(v) \cup \text{in}(v))
\end{align*}

\text{return } (V, A')

- Timing: \( O(V + A) \)
- Quality guarantee: \( |A'| \geq \)
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph \( D = (V, A) \))

\[ A' \leftarrow \emptyset \text{ (these will be the edges we keep)} \]

\textbf{foreach} \( v \in V \) \textbf{do}

\textbf{if} \( \text{outdeg}(v) > \text{indeg}(v) \) \textbf{then}

\[ A' \leftarrow A' \cup \text{out}(v) \]

\textbf{else}

\[ A' \leftarrow A' \cup \text{in}(v) \]

\[ A \leftarrow A \setminus (\text{out}(v) \cup \text{in}(v)) \]

\textbf{return} \((V, A')\)

- \textbf{Timing:} \( O(V + A) \)
- \textbf{Quality guarantee:} \( |A'| \geq |A|/2 \)
Improved Greedy-Heuristic for FAS

- Each iteration of `foreach`, first look for sources and sinks. If there are none, pick $v$ such that $|\text{outdeg}(v) - \text{indeg}(v)|$ is maximized.
Improved Greedy-Heuristic for FAS

• Each iteration of `foreach`, first look for sources and sinks. If there are none, pick $v$ such that $|\text{outdeg}(v) - \text{indeg}(v)|$ is maximized.

• Timing: ?
Improved Greedy-Heuristic for FAS

- Each iteration of `foreach`, first look for sources and sinks. If there are none, pick $v$ such that $|\text{outdeg}(v) - \text{indeg}(v)|$ is maximized.

- Timing: $O(V + A)$
  
  maintain partition of vertices ordered by $|\text{outdeg}(v) - \text{indeg}(v)|$. 


Improved Greedy-Heuristic for FAS

- Each iteration of `foreach`, first look for sources and sinks. If there are none, pick \( v \) such that \( |\text{outdeg}(v) - \text{indeg}(v)| \) is maximized.

- Timing: \( O(V + A) \)
  
  - maintain partition of vertices ordered by \( |\text{outdeg}(v) - \text{indeg}(v)| \).

- Quality guarantee: \( |A'| \geq \)
Improved Greedy-Heuristic for FAS

- Each iteration of `foreach`, first look for sources and sinks. If there are none, pick $v$ such that $|\text{outdeg}(v) - \text{indeg}(v)|$ is maximized.

- Timing: $O(V + A)$ maintain partition of vertices ordered by $|\text{outdeg}(v) - \text{indeg}(v)|$.

- Quality guarantee: $|A'| \geq |A|/2 + |V|/6$
Improved Greedy-Heuristic for FAS

- Each iteration of `foreach`, first look for sources and sinks. If there are none, pick $v$ such that $|\text{outdeg}(v) - \text{indeg}(v)|$ is maximized.

- Timing: $O(V + A)$

- Quality guarantee: $|A'| \geq |A|/2 + |V|/6$
Improved Greedy-Heuristic for FAS

- Each iteration of `foreach`, first look for sources and sinks. If there are none, pick $v$ such that $|\text{outdeg}(v) - \text{indeg}(v)|$ is maximized.

- Timing: $O(V + A)$

- Quality guarantee: $|A'| \geq |A|/2 + |V|/6$
Step 2: Leveling
Step 2: Leveling

Problem

- **Input:** acyclic, directed graph \( D = (V, A) \)
- **Output:**
Step 2: Leveling

Problem

- **Input:** acyclic, directed graph $D = (V, A)$
- **Output:** Mapping $y : V \rightarrow \{1, \ldots, |V|\}$, so that for every $uv \in A$, $y(u) < y(v)$. 
Step 2: Leveling

Problem

- Input: acyclic, directed graph $D = (V, A)$
- Output: Mapping $y : V \rightarrow \{1, \ldots, |V|\}$, so that for every $uv \in A$, $y(u) < y(v)$.

Objective: minimize . . .
Step 2: Leveling

Problem

- **Input:** acyclic, directed graph $D = (V, A)$
- **Output:** Mapping $y: V \rightarrow \{1, \ldots, |V|\}$, so that for every $uv \in A$, $y(u) < y(v)$.

Objective: *minimize* . . .

- Number of layers, i.e. $|y(V)|$
- Length of the longest edge, i.e. $\max_{uv \in A} y(v) - y(u)$
- Total edge length, i.e. number of dummy vertices
Step 2: Leveling

Problem

- **Input:** acyclic, directed graph \( D = (V, A) \)
- **Output:** Mapping \( y : V \to \{1, \ldots, |V|\} \),
  so that for every \( uv \in A \), \( y(u) < y(v) \).

Objective: *minimize* . . .

- **Number of layers**, i.e. \( |y(V)| \)
- **Length of the longest edge**, i.e. \( \max_{uv \in A} y(v) - y(u) \)
- **Total edge length**, i.e. number of dummy vertices
Algorithm to Minimize the Number of Layers
Algorithm to Minimize the Number of Layers

- for each source $q$
  set $y(q) := 1$
Algorithm to Minimize the Number of Layers

- for each source $q$
  set $y(q) := 1$
- for each non-source $v$
  set $y(v) := \max \{ y(u) \mid uv \in A \} + 1$
Algorithm to Minimize the Number of Layers

- for each source \( q \)
  set \( y(q) := 1 \)
- for each non-source \( v \)
  set \( y(v) := \max \{ y(u) \mid uv \in A \} + 1 \)

Obs. \( y(v) \) is...
Algorithm to Minimize the Number of Layers

- for each source \( q \)
  
  set \( y(q) := 1 \)

- for each non-source \( v \)
  
  set \( y(v) := \max \{ y(u) \mid uv \in A \} + 1 \)

**Obs.** \( y(v) \) is...

Length of the longest path from a source to \( v \) plus 1.
Algorithm to Minimize the Number of Layers

- for each source $q$
  set $y(q) := 1$

- for each non-source $v$
  set $y(v) := \max \{ y(u) \mid uv \in A \} + 1$

**Obs.** $y(v)$ is... Length of the longest path from a source to $v$ plus 1. ...also optimal with respect to the number of layers!
Algorithm to Minimize the Number of Layers

- for each source \( q \)
  
  set \( y(q) := 1 \)

- for each non-source \( v \)
  
  set \( y(v) := \max \{ y(u) \mid uv \in A \} + 1 \)

**Obs.** \( y(v) \) is...

  Length of the longest path from a source to \( v \) plus 1.

  ...also optimal with respect to the number of layers!

**Question:** Can we do this in linear time?
Linear time implementation

- for each source $q$
  \[ y(q) := 1 \]
- for each non-source $v$
  \[ y(v) := \max\{y(u) \mid uv \in A\} + 1 \]
Linear time implementation

ComputeLayering(AcyclicDigraph \(D = (V, A)\))

\[
y = \text{new int}[1..|V|] \quad // \text{all} \quad == \quad 0
\]

\begin{itemize}
  \item foreach source \(q \in V\) do
    \[
      y(q) \leftarrow 1
    \]
  \item foreach non-source \(v \in V\) do
    \[
      \text{ComputeYRec}(D, v, y)
    \]
\end{itemize}

return \(y\)
Linear time implementation

ComputeLayering(AcyclicDigraph $D = (V, A))$

```java
y = new int[1..|V|]  // all == 0
foreach source $q \in V$ do
  $y(q) \leftarrow 1$
foreach non-source $v \in V$ do
  ComputeYRec($D, v, y$)
return $y$
```

ComputeYRec(AcyclicDigraph $D = (V, A)$, Vertex $v$, int[] $y$)

```java
if $y(v) == 0$ then
  $y(v) \leftarrow$
return $y(v)$
```
Linear time implementation

ComputeLayering(AcyclicDigraph $D = (V, A)$)

\[
y = \text{new int}[1..|V|] \quad // \quad \text{all} = 0
\]

\[\text{foreach source } q \in V \quad \text{do}\]
\[\quad y(q) \leftarrow 1\]

\[\text{foreach non-source } v \in V \quad \text{do}\]
\[\quad \text{ComputeYRec}(D, v, y)\]

\text{return } y

ComputeYRec(AcyclicDigraph $D = (V, A)$, Vertex $v$, int[] $y$)

\[\text{if } y(v) == 0 \text{ then}\]
\[\quad y(v) \leftarrow \max \{\text{ComputeYRec}(D, u, y) \mid uv \in A\} + 1\]

\text{return } y(v)
Our Example
Our Example
Our Example

Looks good .... right?
Our Example

Looks good .... right?
Our Example

Looks good .... right?
Our Example

Looks good .... right?

The drawing can be very wide :-(

Steven Chaplick · Lehrstuhl für Informatik I · Universität Würzburg
Goal: Narrower layer assignment.

Problem: Leveling with a given width.

- **Input:** acyclic, digraph $D = (V, A)$, width $W > 0$
- **Output:** Partition the vertex set into a minimum number of layers such that each layer contains at most $W$ elements.
Goal: Narrower layer assignment.

Problem: Leveling with a given width.
- Input: acyclic, digraph $D = (V, A)$, width $W > 0$
- Output: Partition the vertex set into a minimum number of layers such that each layer contains at most $W$ elements.

Problem: Precedence-Constrained Multi-Processor Scheduling
- Input: $n$ jobs with unit (1) processing time, $W$ identical machines, and a partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ and having minimum processing time.
Goal: Narrower layer assignment.

Problem: Leveling with a given width.

- Input: acyclic, digraph $D = (V, A)$, width $W > 0$
- Output: Partition the vertex set into a minimum number of layers such that each layer contains at most $W$ elements.

Problem: Precedence-Constrained Multi-Processor Scheduling

- Input: $n$ jobs with unit (1) processing time, $W$ identical machines, and a partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ and having minimum processing time.
- NP-hard, $(2 - \frac{2}{W})$-Approx., no $(\frac{4}{3} - \varepsilon)$-Approx. ($W \geq 3$).
Goal: Narrower layer assignment.

Problem: Leveling with a given width.

- **Input:** acyclic, digraph $D = (V, A)$, width $W > 0$
- **Output:** Partition the vertex set into a minimum number of layers such that each layer contains at most $W$ elements.

Problem: Precedence-Constrained Multi-Processor Scheduling

- **Input:** $n$ jobs with unit (1) processing time, $W$ identical machines, and a partial ordering $<$ on the jobs.
- **Output:** Schedule respecting $<$ and having minimum processing time.
- **NP-hard,** $(2 - \frac{2}{W})$-Approx., no $(\frac{4}{3} - \varepsilon)$-Approx. ($W \geq 3$).
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

![Precedence graph](image)
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

Number of Machines is \( W = 2 \).
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

Number of Machines is $W = 2$.

**Output:** Schedule
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

Number of Machines is $W = 2$.

**Output:** Schedule

| $M_1$ |  
|-------|---
| $M_2$ | 
| $t$   | 1 2 3 4 5 6 7 8 9 10  |

Diagram:

```
1 -> 3 -> 5      6 -> 7     9 -> C
|     |     |     |
2    4    5    8    9  C    E
                   |
                   B
                   |
                   D
                 
                   F
                   |
                   G
```
Approximating PCMPS

**Input:**
Precedence graph (divided into layers of arbitrary width)

```
1 → 2 → 3 → 4
  |    |    |
  5 → 6 → 7 → 8 → 9
     |    |    |
    A → B → C → D → E
      |    |    |
       |    |    |
        |    |    |
         F → G
```

Number of Machines is $W = 2$.

**Output:**
Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>-</td>
</tr>
<tr>
<td>$t$</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>
Approximating PCMPS

Input: Precedence graph (divided into layers of arbitrary width)

Number of Machines is $W = 2$.

Output: Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Approximating PCMPS

**Input:**  
Precedence graph (divided into layers of arbitrary width)

```
1 -- 2  
|   |   
3   5
|   |   
4 --
```

Number of Machines is $W = 2$.

**Output:**  
Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

![Precedence graph](attachment:image.png)

Number of Machines is $W = 2$.

**Output:** Schedule

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 4 5</td>
<td>- 3 -</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

- $M_1$: Tasks scheduled on machine $M_1$.
- $M_2$: Tasks scheduled on machine $M_2$.
- $t$: Time steps.
Approximating PCMPS

**Input:**  
Precedence graph (divided into layers of arbitrary width)

```
\begin{figure}[h]
\centering
\begin{tikzpicture}
\node (1) at (0,0) {1};
\node (2) at (1,1) {2};
\node (3) at (1,-1) {3};
\node (4) at (2,-1) {4};
\node (5) at (3,0) {5};
\node (6) at (3,1) {6};
\node (7) at (4,-1) {7};
\node (8) at (5,0) {8};
\node (9) at (5,1) {9};
\node (10) at (6,-1) {10};
\node (11) at (7,0) {A};
\node (12) at (7,1) {B};
\node (13) at (8,-1) {C};
\node (14) at (9,0) {D};
\node (15) at (9,1) {E};
\node (16) at (10,-1) {F};
\node (17) at (11,0) {G};
\draw[->] (1) -- (2);
\draw[->] (1) -- (3);
\draw[->] (1) -- (4);
\draw[->] (2) -- (5);
\draw[->] (3) -- (5);
\draw[->] (4) -- (5);
\draw[->] (5) -- (6);
\draw[->] (5) -- (7);
\draw[->] (6) -- (8);
\draw[->] (7) -- (8);
\draw[->] (8) -- (9);
\draw[->] (9) -- (11);
\draw[->] (9) -- (12);
\draw[->] (11) -- (13);
\draw[->] (12) -- (14);
\draw[->] (11) -- (15);
\draw[->] (12) -- (15);
\draw[->] (15) -- (16);
\draw[->] (15) -- (17);
\end{tikzpicture}
\end{figure}
```

Number of Machines is $W = 2$.

**Output:**  
Schedule

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 4 5 6</td>
<td>- 3 - - 7</td>
</tr>
<tr>
<td>$t$</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>
Approximating PCMPS

Input: Precedence graph (divided into layers of arbitrary width)

Number of Machines is $W = 2$.

Output: Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1 2 4 5 6 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>– 3 – – 7 9</td>
</tr>
<tr>
<td>$t$</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

![Precedence graph](image)

Number of Machines is \( W = 2 \).

**Output:** Schedule

<table>
<thead>
<tr>
<th>( M_1 )</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_2 )</td>
<td>–</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>7</td>
<td>9</td>
<td>B</td>
</tr>
<tr>
<td>( t )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

![Graph with nodes and arrows]

Number of Machines is $W = 2$.

**Output:** Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

Input graph:

```
1 → 3 → 5 → 6 → 7 → A → B → C → D → E → F → G
```

Number of Machines is \( W = 2 \).

**Output:** Schedule

<table>
<thead>
<tr>
<th></th>
<th>( M_1 )</th>
<th>( M_2 )</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 4 5 6 8 A C E</td>
<td>– 3 – – 7 9 B D F</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>
Approximating PCMPS

**Input:**
Precedence graph (divided into layers of arbitrary width)

Number of Machines is $W = 2$.

**Output:**
Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
Approximating PCMPS

Input: Precedence graph (divided into layers of arbitrary width)

\[ \begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
M_1 & 1 & 2 & 4 & 5 & 6 & 8 & A & C & E & G \\
M_2 & - & 3 & - & - & 7 & 9 & B & D & F & - \\
t & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{array} \]

Number of Machines is \( W = 2 \).

Output: Schedule

Question: Good approximation factor?
Algorithm

• jobs stored in a list $L$
  (in any order, e.g., topologically sorted).

• for each time $t = 1, 2, \ldots$ schedule $\leq W$ available jobs.

• a job in $L$ is available when all its predecessors have been scheduled.

• As long as there are free machines and available jobs, take the first available job and assign it to a free machine.
Analysis for $W = 2$

**Precedence graph** $G_{<}$

```
1  2  3  4  5  6  7  8  9  10
A  B  C  D  E  F  G
```

**Schedule**

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

„The art of the lower bound“
Analysis for $W = 2$

Precedence graph $G_<$

```
1 2 3 4 5 6 7 8 9
\rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow
A B C D E F G
```

Schedule

```
\begin{array}{l|cccccccccc}
M_1 & 1 & 2 & 4 & 5 & 6 & 8 & A & C & E & G \\
M_2 & - & 3 & - & - & 7 & 9 & B & D & F & - \\
t & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
```

„The art of the lower bound“

OPT $\geq$
Analysis for $W = 2$

"The art of the lower bound"

\[ \text{OPT} \geq \left\lceil \frac{n}{2} \right\rceil \]
Analysis for $W = 2$

```

```

Precedence graph $G_<$

Schedule

```

„The art of the lower bound“

$\text{OPT} \geq \lceil n/2 \rceil$ and $\text{OPT} \geq$
Analysis for $W = 2$

Precedence graph $G_<$

Schedule

$OPT \geq \lceil n/2 \rceil$ and $OPT \geq \ell := \text{Number of layers of } G_<$
Analysis for $W = 2$

Precedence graph $G_<$

Schedule

```
M_1 | 1 2 4 5 6 8 A C E G
M_2 | - 3 - - 7 9 B D F -
```

```
t | 1 2 3 4 5 6 7 8 9 10
```

„The art of the lower bound“

$\text{OPT} \geq \lceil n/2 \rceil$ and $\text{OPT} \geq \ell := \text{Number of layers of } G_<$

**Goal:** measure the quality of our algorithm using the lower bound(s).
Analysis for $W = 2$

Precedence graph $G_<$

Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

"The art of the lower bound"

$\text{OPT} \geq \lceil n/2 \rceil$ and $\text{OPT} \geq \ell := \text{Number of layers of } G_<$

Goal: measure the quality of our algorithm using the lower bound(s).

Bound $\text{ALG} \leq$
Analysis for $W = 2$

Precedence graph $G_<$

```
1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → C → D → E → F → G
```

Schedule

```
<table>
<thead>
<tr>
<th>M_1</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_2</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
```

"The art of the lower bound"

$\text{OPT} \geq \lceil n/2 \rceil$ and $\text{OPT} \geq \ell := \text{Number of layers of } G_<$

**Goal:** measure the quality of our algorithm using the lower bound(s).

**Bound** $\text{ALG} \leq$
Analysis for $W = 2$

Precedence graph $G <$

Schedule

```
\begin{array}{ccccccccccc}
M_1 & 1 & 2 & 4 & 5 & 6 & 8 & A & C & E & G \\
M_2 & - & 3 & - & - & 7 & 9 & B & D & F & - \\
t & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
```

„The art of the lower bound“

OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G <$

Goal: measure the quality of our algorithm using the lower bound(s).

Bound $\text{ALG} \leq$

insertion of pauses (−) in the schedule (except the last) maps to layers of $G <$
Analysis for $W = 2$

Precedence graph $G_<$

Schedule

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>G</td>
</tr>
<tr>
<td>$M_2$</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>9</td>
<td></td>
<td>B</td>
<td>D</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
```

„The art of the lower bound“

$\text{OPT} \geq \lceil n/2 \rceil$ and $\text{OPT} \geq \ell := \text{Number of layers of } G_<$

Goal: measure the quality of our algorithm using the lower bound(s).

Bound $\text{ALG} \leq \left\lceil \frac{n+\ell}{2} \right\rceil$

insertion of pauses ( ) in the schedule (except the last) maps to layers of $G_<$
Analysis for $W = 2$

Precedence graph $G_<$

Schedule

"The art of the lower bound"

OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell := \text{Number of layers of } G_<$

Goal: measure the quality of our algorithm using the lower bound(s).

Bound $\text{ALG} \leq \left\lceil \frac{n + \ell}{2} \right\rceil \approx$

insertion of pauses (−) in the schedule (except the last) maps to layers of $G_<$
Analysis for $W = 2$

Precedence graph $G_<$

Schedule

```
| $M_1$ | 1 2 4 5 6 8 A C E G |
| $M_2$ | 3 7 9 B D F - |
| $t$   | 1 2 3 4 5 6 7 8 9 10 |
```

„The art of the lower bound“

OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell := \text{Number of layers of } G_<$

**Goal:** measure the quality of our algorithm using the lower bound(s).

**Bound** $\text{ALG} \leq \lceil \frac{n+\ell}{2} \rceil \approx \lceil n/2 \rceil + \ell/2$

insertion of pauses (−) in the schedule (except the last) maps to layers of $G_<$
Analysis for $W = 2$

**Precedence graph** $G_<$

```
1  3  5  6  9
2  4  7  8  C
          E
          F
          G
```

**Schedule**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
```

"The art of the lower bound"

OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G_<$

**Goal:** measure the quality of our algorithm using the lower bound(s).

**Bound** $\text{ALG} \leq \left\lfloor \frac{n + \ell}{2} \right\rfloor \approx \lceil n/2 \rceil + \ell/2 \leq$

insertion of pauses (−) in the schedule (except the last) maps to layers of $G_<$
Analysis for $W = 2$

Precedence graph $G_<$

Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td></td>
<td>-</td>
<td>3</td>
<td>-</td>
<td></td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

„The art of the lower bound“

$\text{OPT} \geq \lceil n/2 \rceil$ and \ $\text{OPT} \geq \ell := \text{Number of layers of } G_<$

Goal: measure the quality of our algorithm using the lower bound(s).

Bound $\text{ALG} \leq \left\lceil \frac{n + \ell}{2} \right\rceil \approx \left\lceil n/2 \right\rceil + \ell/2 \leq 3/2 \cdot \text{OPT}$

insertion of pauses (−) in the schedule (except the last) maps to layers of $G_<$
Analysis for $W = 2$

**Precedence graph $G_<$**

```
1 → 2 → 5 → 6 → 8
3 → 4 → 7 → 9 → C
```

**Schedule**

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

„The art of the lower bound“

OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell := \text{Number of layers of } G_<$

**Goal:** measure the quality of our algorithm using the lower bound(s).

Gen. $\leq (2 - 1/W) \cdot \text{OPT}$

**Bound** $\text{ALG} \leq \left\lceil \frac{n + \ell}{2} \right\rceil \approx \left\lceil n/2 \right\rceil + \ell/2 \leq 3/2 \cdot \text{OPT}$

insertion of pauses (−) in the schedule (except the last) maps to layers of $G_<$
Step 3: Crossing minimization
Step 3: Crossing minimization

Problem:

- Input:
- Output:
Step 3: Crossing minimization

Problem:
- Input: Graph $G$, layering $y : V \rightarrow \{1, \ldots, |V|\}$
- Output:
Step 3: Crossing minimization

Problem:

- **Input:** Graph $G$, layering $y: V \rightarrow \{1, \ldots, |V|\}$
- **Output:** (Re-)ordering of vertices in each layer so that the number of crossings in minimized.
Step 3: Crossing minimization

Problem:

- **Input:** Graph $G$, layering $y: V \rightarrow \{1, \ldots, |V|\}$
- **Output:** (Re-)ordering of vertices in each layer so that the number of crossings in minimized.

- NP-hard, even for 2 layers [Garey & Johnson '83]
- hardly any approaches optimize over multiple layers :-(

Iterative crossing reduction – idea
Iterative crossing reduction – idea

- add dummy-vertices for edges connecting *far* layers.
- consider adjacent layers \((L_1, L_2), (L_2, L_3), \ldots\) bottom-to-top.
- minimize crossings by permuting \(L_{i+1}\) while keeping \(L_i\) fixed.
Iterative crossing reduction – idea

- add dummy-vertices for edges connecting *far* layers.
- consider adjacent layers \((L_1, L_2), (L_2, L_3), \ldots\)
  bottom-to-top.
- minimize crossings by permuting \(L_{i+1}\) while keeping \(L_i\)
  fixed.

**Obs.** The number of crossings only depends on permutations
of adjacent layers.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$. 

(2) iteratively consider adjacent layers $L_i$ and $L_{i+1}$.

(3) minimize crossings by permuting $L_{i+1}$ and keeping $L_i$ fixed.

(4) repeat steps (2)–(3) in the reverse order (starting from $L_h$).

(5) repeat steps (2)–(4) until no further improvement is achieved.

(6) repeat steps (1)–(5) with different starting permutations.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$.

(2) iteratively consider adjacent layers $L_i$ und $L_{i+1}$.
Iterative crossing reduction – Algorithm

1. choose a random permutation of $L_1$.

2. iteratively consider adjacent layers $L_i$ und $L_{i+1}$.

3. minimize crossings by permuting $L_{i+1}$ and keeping ($L_i$ fixed).

4. repeat steps (2)–(3) in the reverse order (starting from $L_h$).

5. repeat steps (2)–(4) until no further improvement is achieved.

6. repeat steps (1)–(5) with different starting permutations.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$.

(2) iteratively consider adjacent layers $L_i$ und $L_{i+1}$.

(3) minimize crossings by permuting $L_{i+1}$ and keeping ($L_i$ fixed).

(4) repeat steps (2)–(3) in the reverse order (starting from $L_h$).

(5) repeat steps (2)–(4) until no further improvement is achieved.

(6) repeat steps (1)–(5) with different starting permutations.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$.

(2) iteratively consider adjacent layers $L_i$ und $L_{i+1}$.

(3) minimize crossings by permuting $L_{i+1}$ and keeping ($L_i$ fixed).

(4) repeat steps (2)–(3) in the reverse order (starting from $L_h$).

(5) repeat steps (2)–(4) until no further improvement is achieved.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$.

(2) iteratively consider adjacent layers $L_i$ und $L_{i+1}$.

(3) minimize crossings by permuting $L_{i+1}$ and keeping ($L_i$ fixed).

(4) repeat steps (2)–(3) in the reverse order (starting from $L_h$).

(5) repeat steps (2)–(4) until no further improvement is achieved.

(6) repeat steps (1)–(5) with different starting permutations.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$.

(2) iteratively consider adjacent layers $L_i$ und $L_{i+1}$.

(3) minimize crossings by permuting $L_{i+1}$ and keeping ($L_i$ fixed).

(4) repeat steps (2)–(3) in the reverse order (starting from $L_h$).

(5) repeat steps (2)–(4) until no further improvement is achieved.

(6) repeat steps (1)–(5) with different starting permutations.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$.

(2) iteratively consider adjacent layers $L_i$ und $L_{i+1}$.

(3) minimize crossings by permuting $L_{i+1}$ and keeping ($L_i$ fixed).

(4) repeat steps (2)–(3) in the reverse order (starting from $L_h$).

(5) repeat steps (2)–(4) until no further improvement is achieved.

(6) repeat steps (1)–(5) with different starting permutations.
One-sided Crossing Minimization

Problem

• Input:

• Output:
One-sided Crossing Minimization

Problem

- **Input:** bipartite graph $G = (L_1 \cup L_2, E)$, permutation $\pi_1$ on $L_1$
- **Output:**

Diagram: [Image of a bipartite graph with vertices and edges labeled 4 to 30]
One-sided Crossing Minimization

Problem

• **Input:** bipartite graph \( G = (L_1 \cup L_2, E) \), permutation \( \pi_1 \) on \( L_1 \)

• **Output:** permutation \( \pi_2 \) von \( L_2 \), the number of edge crossings is minimized.
One-sided Crossing Minimization

Problem

- **Input:** bipartite graph \( G = (L_1 \cup L_2, E) \),
  permutation \( \pi_1 \) on \( L_1 \)
- **Output:** permutation \( \pi_2 \) von \( L_2 \), the number of edge crossings is minimized.

One-sided crossing minimization is NP-hard.
One-sided Crossing Minimization

Problem

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation $\pi_1$ on $L_1$
- Output: permutation $\pi_2$ von $L_2$, the number of edge crossings is minimized.

One-sided crossing minimization is NP-hard.

Algorithms

- barycenter heuristic
- median heuristic
- Greedy-Switch
- ILP
Barycentre Heuristic

• Intuition: few intersections occur when vertices are close to their neighbours
Barycentre Heuristic

- Intuition: few intersections occur when vertices are close to their neighbours
- The barycentre of \( u \) is the average \( x \)-coordinate of the neighbours of \( u \) in layer \( L_1 \) 

\[
x_2(u) := \text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v)
\]
Barycentre Heuristic

- Intuition: few intersections occur when vertices are close to their neighbours
- The barycentre of $u$ is the average $x$-coordinate of the neighbours of $u$ in layer $L_1$: $[x_1 \equiv \pi_1]$

$$x_2(u) := \text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v)$$

- Vertices with the same barycentre of are offset by a small $\delta$
Barycentre Heuristic [Sugiyama et al. ’81]

- Intuition: few intersections occur when vertices are close to their neighbours.
- The barycentre of $u$ is the average $x$-coordinate of the neighbours of $u$ in layer $L_1$:

\[ x_2(u) := bary(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v) \]

- Vertices with the same barycentre of $u$ are offset by a small $\delta$.
- Linear runtime.
- Relatively good results.
- Optimal if no crossings are required.
- $O(\sqrt{n})$-approximation factor.
Barycentre Heuristic

- Intuition: few intersections occur when vertices are close to their neighbours
- The barycentre of $u$ is the average $x$-coordinate of the neighbours of $u$ in layer $L_1$  
  
  $x_2(u) := \text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v)$

- vertices with the same barycentre of are offset by a small $\delta$
- linear runtime
- relatively good results
- optimal if no crossings are required  
  
  $O(\sqrt{n})$-approximation factor
Barycentre Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbours.
- The barycentre of \( u \) is the average \( x \)-coordinate of the neighbours of \( u \) in layer \( L_1 \) \( [x_1 \equiv \pi_1] \)

\[
x_2(u) := \text{bary}(u) := \frac{1}{\text{deg}(u)} \sum_{v \in N(u)} x_1(v)
\]

- vertices with the same barycentre of are offset by a small \( \delta \)
- linear runtime
- relatively good results
- optimal if no crossings are required \( \text{exercise!} \)
- \( O(\sqrt{n}) \)-approximation factor
Barycentre Heuristic

[Sugiyama et al. ’81]

- Intuition: few intersections occur when vertices are close to their neighbours
- The barycentre of $u$ is the average $x$-coordinate of the neighbours of $u$ in layer $L_1$ $[x_1 \equiv \pi_1]$

$$x_2(u) := \text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v)$$

- vertices with the same barycentre of are offset by a small $\delta$
- linear runtime
- relatively good results
- optimal if no crossings are required \(\text{exercise!}\)
- $O(\sqrt{n})$-approximation factor
Barycentre Heuristic

- Intuition: few intersections occur when vertices are close to their neighbours
- The barycentre of $u$ is the average $x$-coordinate of the neighbours of $u$ in layer $L_1$ \[ x_1 \equiv \pi_1 \]
  \[
x_2(u) := \text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v)
\]
- vertices with the same barycentre of are offset by a small $\delta$
- linear runtime
- relatively good results
- optimal if no crossings are required \( \leftarrow \text{exercise!} \)
- $O(\sqrt{n})$-approximation factor
Median heuristic

- \{v_1, \ldots, v_k\} := N(u) with \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)
- 
  \[x_2(u) := \text{med}(u) := \begin{cases} 
  0 & \text{when } N(u) = \emptyset \\
  \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise}
\end{cases}\]

- move vertices \(u\) und \(v\) by small \(\delta\), when \(x_2(u) = x_2(v)\)
Median heuristic

- \{v_1, \ldots, v_k\} := N(u) \text{ with } \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)

- \[ x_2(u) := \text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases} \]

- move vertices \(u\) and \(v\) by small \(\delta\), when \(x_2(u) = x_2(v)\)

- linear runtime
- relatively good results
- optimal, if no crossings are required
- 3-Approximation factor
Median heuristic

- \{v_1, \ldots, v_k\} := N(u) with \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)

- \[x_2(u) := \text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}\]

- move vertices \(u\) and \(v\) by small \(\delta\), when \(x_2(u) = x_2(v)\)

- linear runtime
- relatively good results
- optimal, if no crossings are required
- 3-Approximation factor

proof in [DETT]
Median heuristic

\[ \{v_1, \ldots, v_k\} \subseteq N(u) \text{ with } \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k) \]

\[ x_2(u) := \text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases} \]

- move vertices \( u \) and \( v \) by small \( \delta \), when \( x_2(u) = x_2(v) \)

- linear runtime
- relatively good results
- optimal, if no crossings are required
- 3-Approximation factor

proof in \([DETT]\)
Median heuristic

- \{v_1, \ldots, v_k\} := N(u) \text{ with } \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)

- \begin{align*}
x_2(u) := \text{med}(u) := \begin{cases} 0 & \text{ when } N(u) = \emptyset \\
\pi_1(v_{\lceil k/2 \rceil}) & \text{ otherwise}
\end{cases}
\end{align*}

- move vertices \( u \) and \( v \) by small \( \delta \), when \( x_2(u) = x_2(v) \)

- linear runtime
- relatively good results
- optimal, if no crossings are required
- 3-Approximation factor

proof in [DETT]
Median heuristic

- \( \{v_1, \ldots, v_k\} := N(u) \) with \( \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k) \)
- \( x_2(u) := \text{med}(u) := \begin{cases} 
0 & \text{when } N(u) = \emptyset \\
\pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise}
\end{cases} \)
- move vertices \( u \) und \( v \) by small \( \delta \), when \( x_2(u) = x_2(v) \)
- linear runtime
- relatively good results
- optimal, if no crossings are required
- 3-Approximation factor

\[ 2k(k+1) + k^2 \quad \text{vs.} \quad (k+1)^2 \]
Greedy-Switch heuristic

- iteratively swap each adjacent node as long as crossings decrease.
- runtime $O(L_2)$ per iteration; at most $|L_2|$ iterations
- suitable as post-processing for other heuristics
Greedy-Switch heuristic

- iteratively swap each adjacent node as long as crossings decrease.
- runtime $O(L^2)$ per iteration; at most $|L^2|$ iterations
- suitable as post-processing for other heuristics
Greedy-Switch heuristic

- iteratively swap each adjacent node as long as crossings decrease.
- runtime $O(L_2)$ per iteration; at most $|L_2|$ iterations
- suitable as post-processing for other heuristics

Worst case?
Greedy-Switch heuristic

- iteratively swap each adjacent node as long as crossings decrease.
- runtime $O(L_2)$ per iteration; at most $|L_2|$ iterations
- suitable as post-processing for other heuristics

**Worst case?**

![Diagram](image_url)
Greedy-Switch heuristic

• iteratively swap each adjacent node as long as crossings decrease.
• runtime $O(L_2)$ per iteration; at most $|L_2|$ iterations
• suitable as post-processing for other heuristics
Greedy-Switch heuristic

- iteratively swap each adjacent node as long as crossings decrease.
- runtime $O(L_2)$ per iteration; at most $|L_2|$ iterations
- suitable as post-processing for other heuristics

Worst case?

\[ \approx k^2/4 \quad \approx 2k \]
Integer Linear Program

- Constant $c_{ij} := \text{num. of crossings between edges incident to } v_i \text{ or } v_j \text{ when } \pi_2(v_i) < \pi_2(v_j)$
Integer Linear Program

- Constant $c_{ij} := \text{num. of crossings between edges incident to } v_i \text{ or } v_j \text{ when } \pi_2(v_i) < \pi_2(v_j)$
- Variable $x_{ij}$ for each $1 \leq i < j \leq n_2 := |L_2|$

\[
x_{ij} = \begin{cases} 
1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\
0 & \text{otherwise}
\end{cases}
\]
Integer Linear Program

- Constant \( c_{ij} := \text{num. of crossings between edges incident to } v_i \text{ or } v_j \text{ when } \pi_2(v_i) < \pi_2(v_j) \)
- Variable \( x_{ij} \) for each \( 1 \leq i < j \leq n_2 := |L_2| \)
  \[ x_{ij} = \begin{cases} 
 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\
 0 & \text{otherwise}
\end{cases} \]
- The number of crossings of a permutations \( \pi_2 \)
  \[ \text{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij} + \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji} \]
  \[ \underbrace{\sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji}}_{\text{constant}} \]
ILP (cont.)

• Minimize the number of crossings:

\[
\text{minimize } \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji}) x_{ij}
\]
ILP (cont.)

- Minimize the number of crossings:

\[
\text{minimize } \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij}
\]

- Constraints:

\[
0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n_2
\]
ILP (cont.)

- Minimize the number of crossings:

\[ \text{minimize} \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij} \]

- Constraints:

\[ 0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n_2 \]

i.e., if \( x_{ij} = 1 \) and \( x_{jk} = 1 \), then \( x_{ik} = 1 \)
ILP (cont.)

- Minimize the number of crossings:

\[
\text{minimize } \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij}
\]

- Constraints:

\[
0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n_2
\]

i.e., if \(x_{ij} = 1\) and \(x_{jk} = 1\), then \(x_{ik} = 0\)

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]
ILP (cont.)

- Minimize the number of crossings:

\[
\text{minimize } \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij}
\]

- Constraints:

\[
0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n_2
\]

i.e., if \(x_{ij} = 1\) and \(x_{jk} = 1\), then \(x_{ik} = 1\)

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

(Transitivity)
ILP (cont.)

- Minimize the number of crossings:
  \[
  \text{minimize } \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij}
  \]

- Constraints:
  
  \[
  0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n_2
  \]
  
  i.e., if \(x_{ij} = 1\) and \(x_{jk} = 1\), then \(x_{ik} = 1\)

Solution can be found via Branch-and-Bound on small degree graphs relatively quickly
Our Example – iterations
Our Example – iterations
Our Example – iterations
Our Example – iterations
Our Example – iterations
Our Example – iterations
Our Example – iterations
Our Example – iterations
Our Example – iterations
Step 4: Vertex positioning

Goal: paths should be close to straight.
Step 4: Vertex positioning

Goal: paths should be close to straight.

Exact: Quadratic Program (QP)

Heuristic: iterative approach
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$

- $x$-coordinate of $v_i$ according to the line $\overline{v_1v_k}$ (with equal spacing):

$$x(v_i) = x(v_1) + \frac{i - 1}{k - 1} (x(v_k) - x(v_1))$$
Quadratic Program

- Consider the path \( p_e = (v_1, \ldots, v_k) \) of an edge \( e = v_1v_k \) with dummy vertices: \( v_2, \ldots, v_{k-1} \)

- \( x \)-coordinate of \( v_i \) according to the line \( \overline{v_1v_k} \) (with equal spacing):

\[
\bar{x}(v_i) = x(v_1) + \frac{i-1}{k-1}(x(v_k) - x(v_1))
\]

- define the deviation from the line

\[
\text{dev}(p_e) := \sum_{i=2}^{k-1} \left( x(v_i) - \bar{x}(v_i) \right)^2
\]
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$

- $x$-coordinate of $v_i$ according to the line $v_1v_k$ (with equal spacing):

  $$x(v_i) = x(v_1) + \frac{i-1}{k-1}(x(v_k) - x(v_1))$$

- Define the deviation from the line

  $$\text{dev}(p_e) := \sum_{i=2}^{k-1} \left( x(v_i) - \overline{x(v_i)} \right)^2$$
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$

- $x$-coordinate of $v_i$ according to the line $\overline{v_1v_k}$ (with equal spacing):

  $$x(v_i) = x(v_1) + \frac{i - 1}{k - 1} (x(v_k) - x(v_1))$$

- Define the deviation from the line

  $$\text{dev}(p_e) := \sum_{i=2}^{k-1} \left( x(v_i) - \overline{x(v_i)} \right)^2$$
QP (cont.)

- Objective function:

\[
\min \sum_{e \in E} \text{dev}(p_e)
\]
QP (cont.)

- Objective function:

\[
\min \sum_{e \in E} \text{dev}(p_e)
\]

- Constraints: for all vertices \(v, w\) in the same layer with \(w\) right of \(v\)

\[
x(w) - x(v) \geq \rho(w, v)
\]
QP (cont.)

- Objective function:

\[
\min \sum_{e \in E} \text{dev}(p_e)
\]

- Constraints: for all vertices \(v, w\) in the same layer with \(w\) right of \(v\)

\[
x(w) - x(v) \geq \rho(w, v)
\]

- \(\rho(w, v)\) is min. horizontal distance between \(w\) and \(v\)
QP (cont.)

- **Objective function:**

\[
\min \sum_{e \in E} \text{dev}(p_e)
\]

- **Constraints:** for all vertices \(v, w\) in the same layer with \(w\) right of \(v\)

\[
x(w) - x(v) \geq \rho(w, v)
\]

- \(\rho(w, v)\) is min. horizontal distance between \(w\) and \(v\)

- **Problem:** QP and potentially exponential width
Iterative Heuristic

- compute an Initial-Layout
Iterative Heuristic

- compute an Initial-Layout
- apply the following steps as long as improvements can be made.
Iterative Heuristic

• compute an Initial-Layout

• apply the following steps as long as improvements can be made.

1. vertex positioning,
2. edge straightening,
3. compactifying the layout width.
Our Example
Our Example
Step 5: drawing the edges
Step 5 – drawing the edges
Step 5 – drawing the edges
Step 5 – drawing the edges

All figs. from [Kaufmann und Wagner: Drawing Graphs]
Our Example
Our Example
Our Example