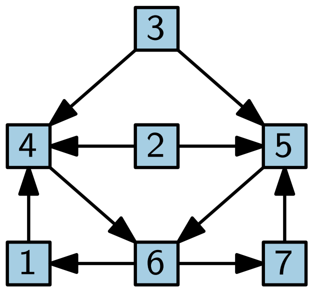
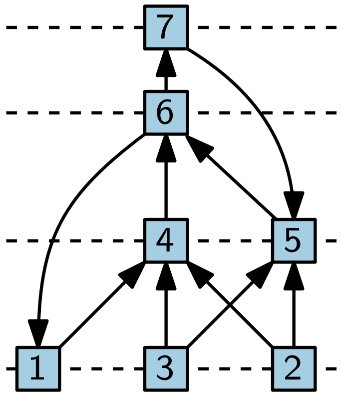
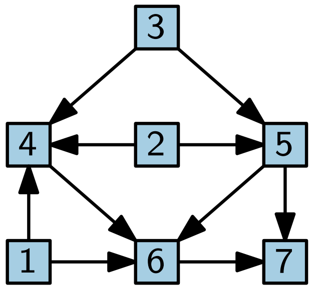
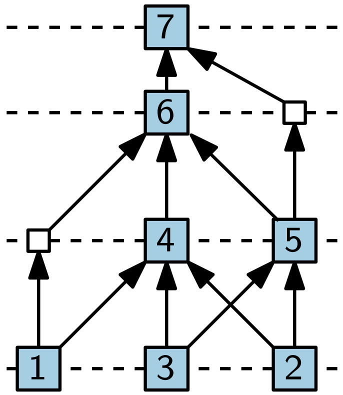


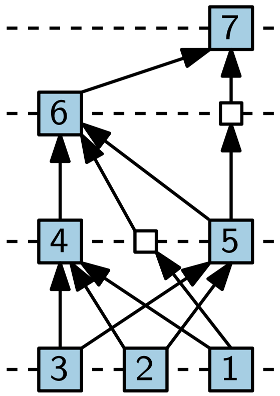


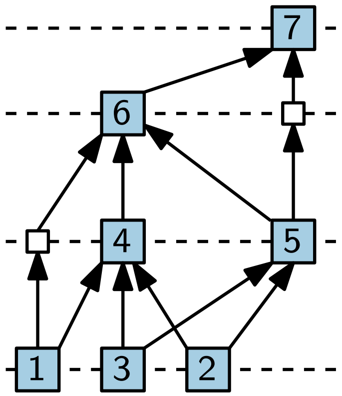
Visualization of Graphs



Lecture 8:
Hierarchical Layouts:
Sugiyama Framework

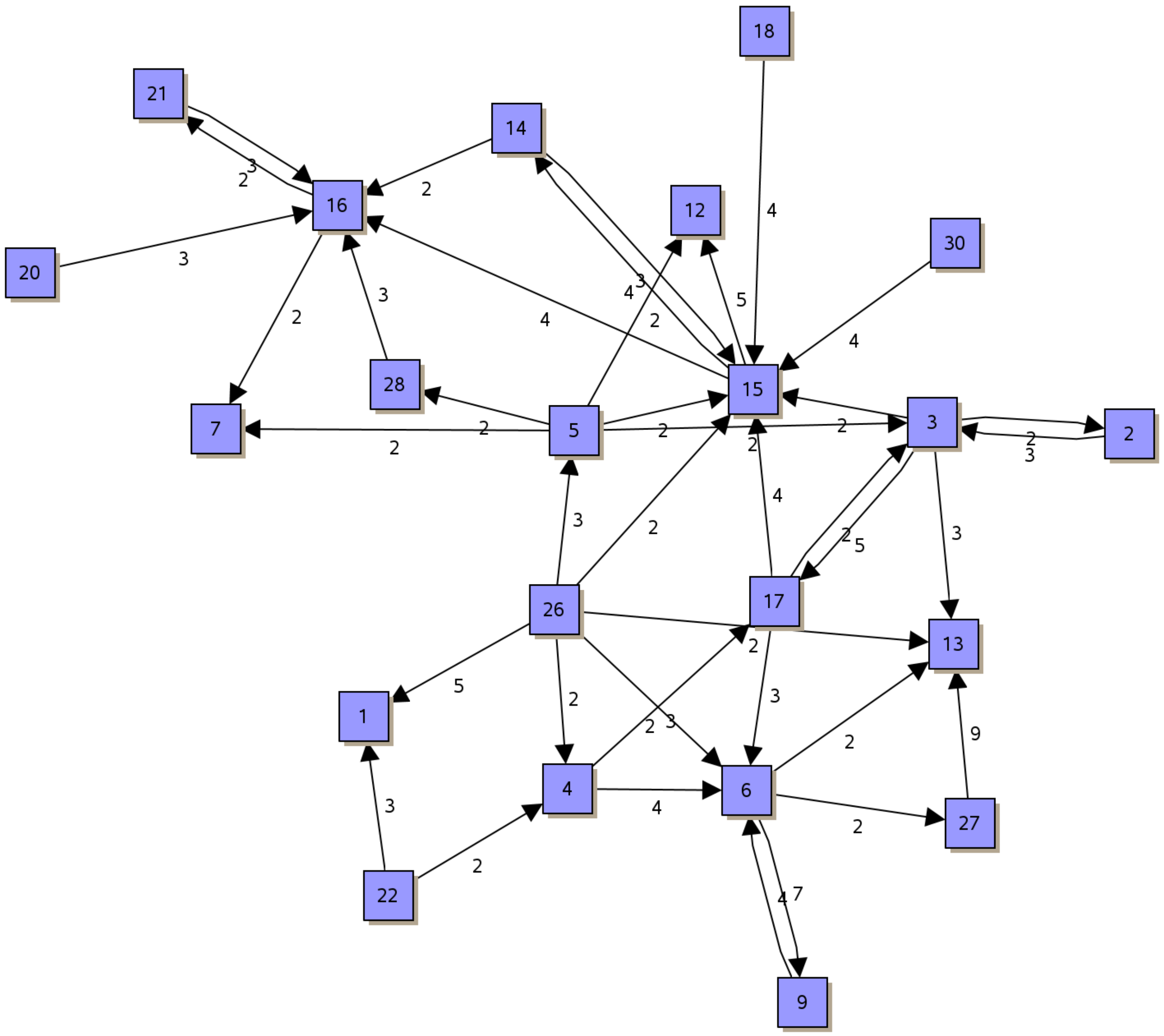




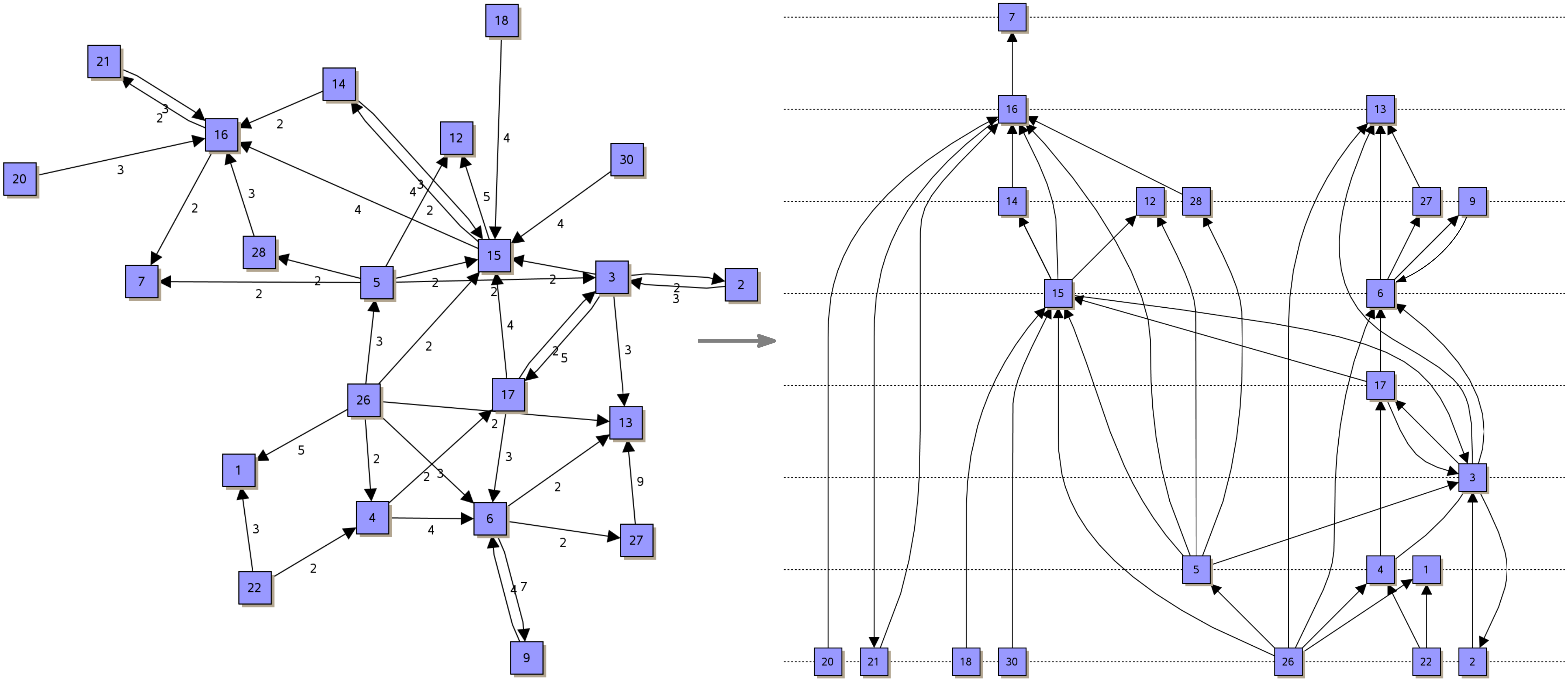


Johannes Zink

Hierarchical Drawings – Motivation



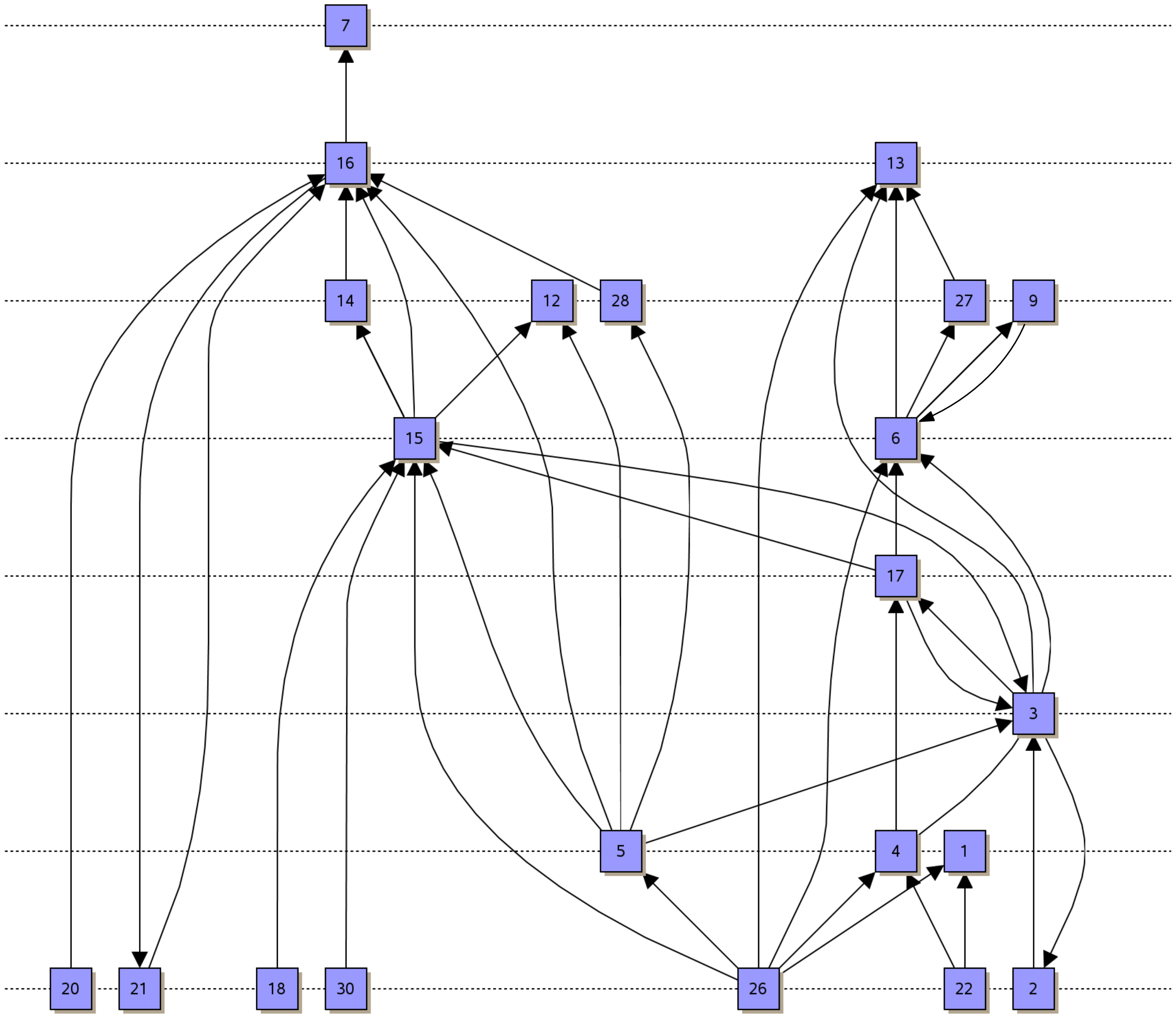
Hierarchical Drawings – Motivation



Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

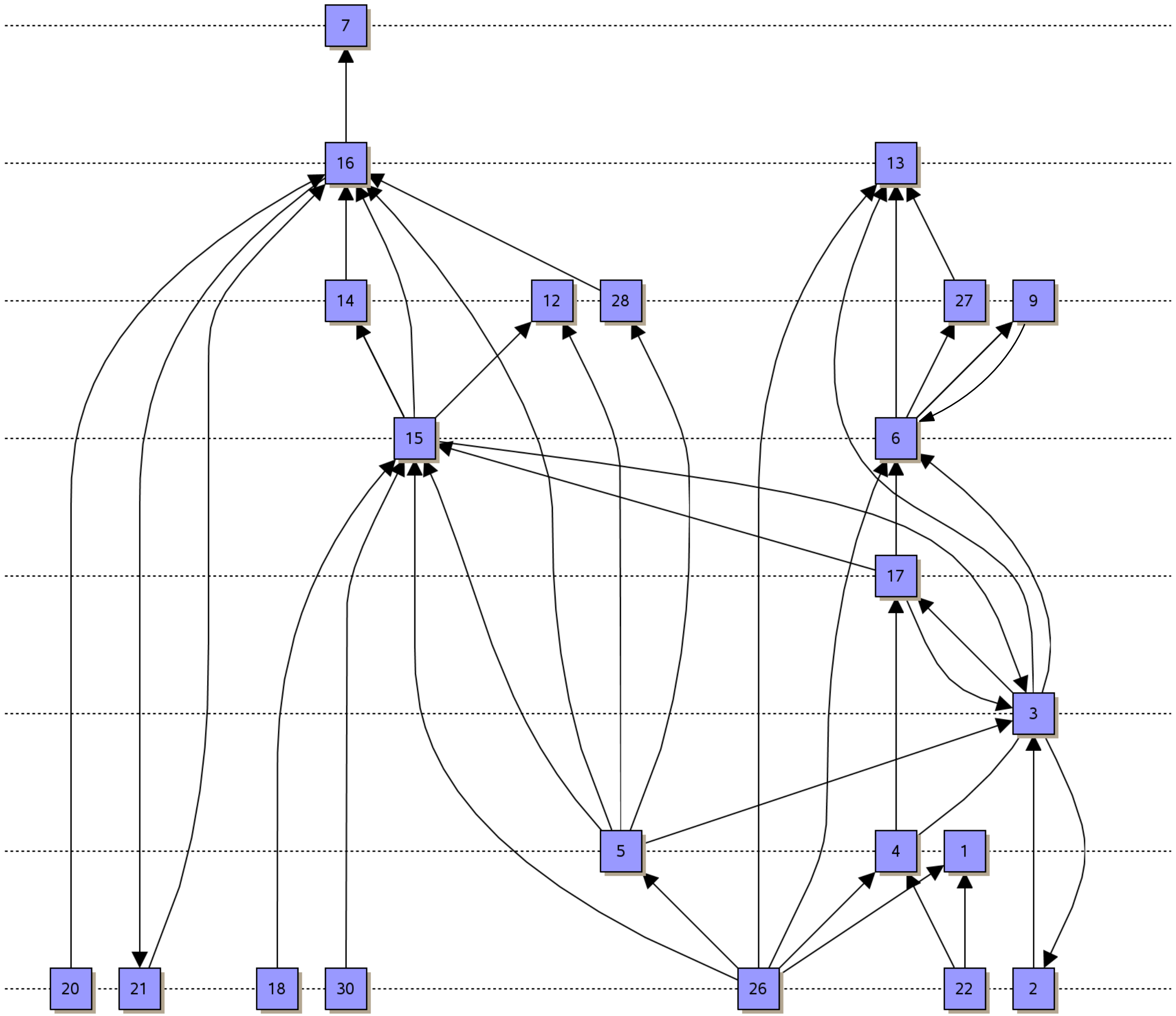


Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties.



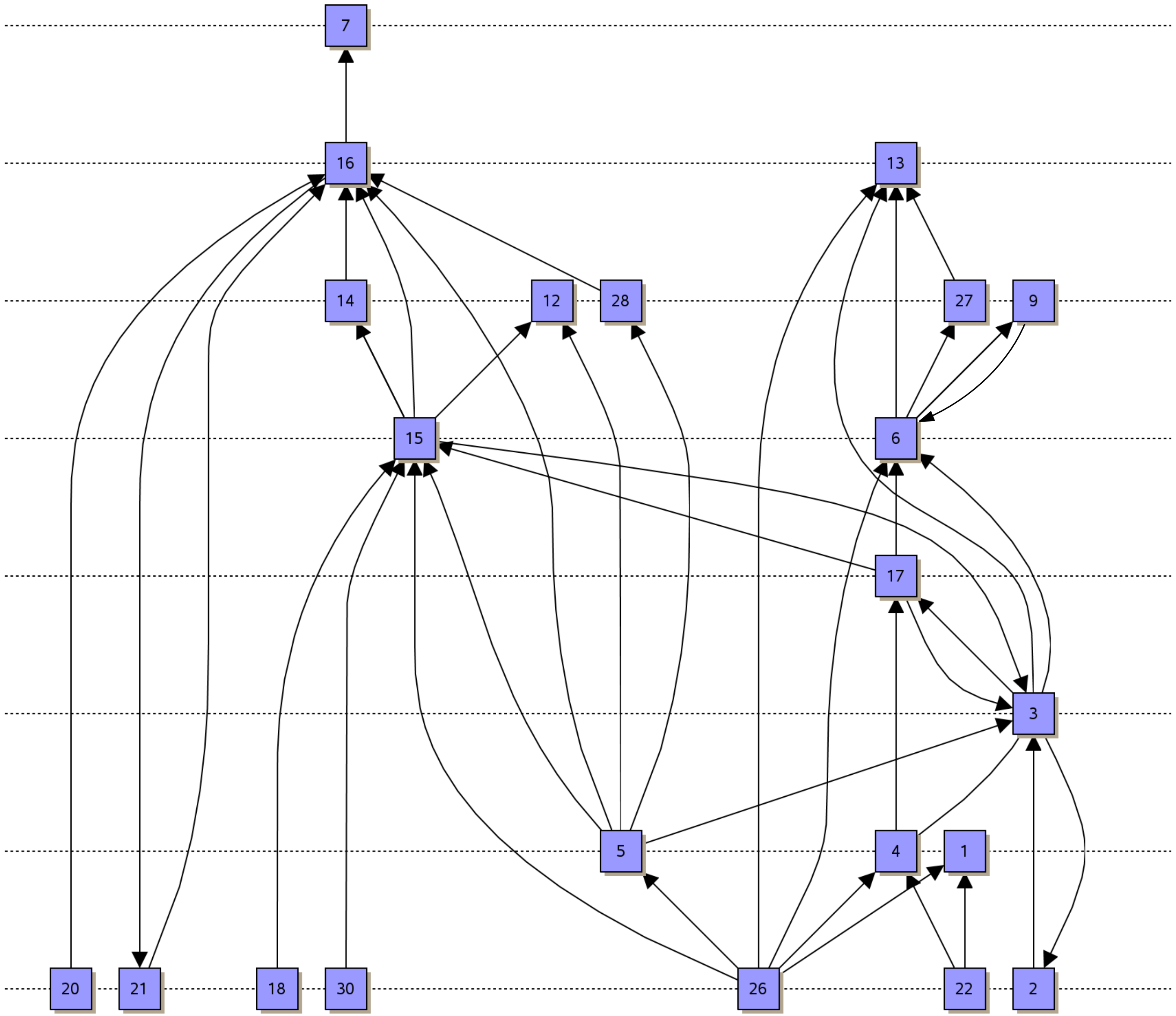
Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties.

- edges directed upwards



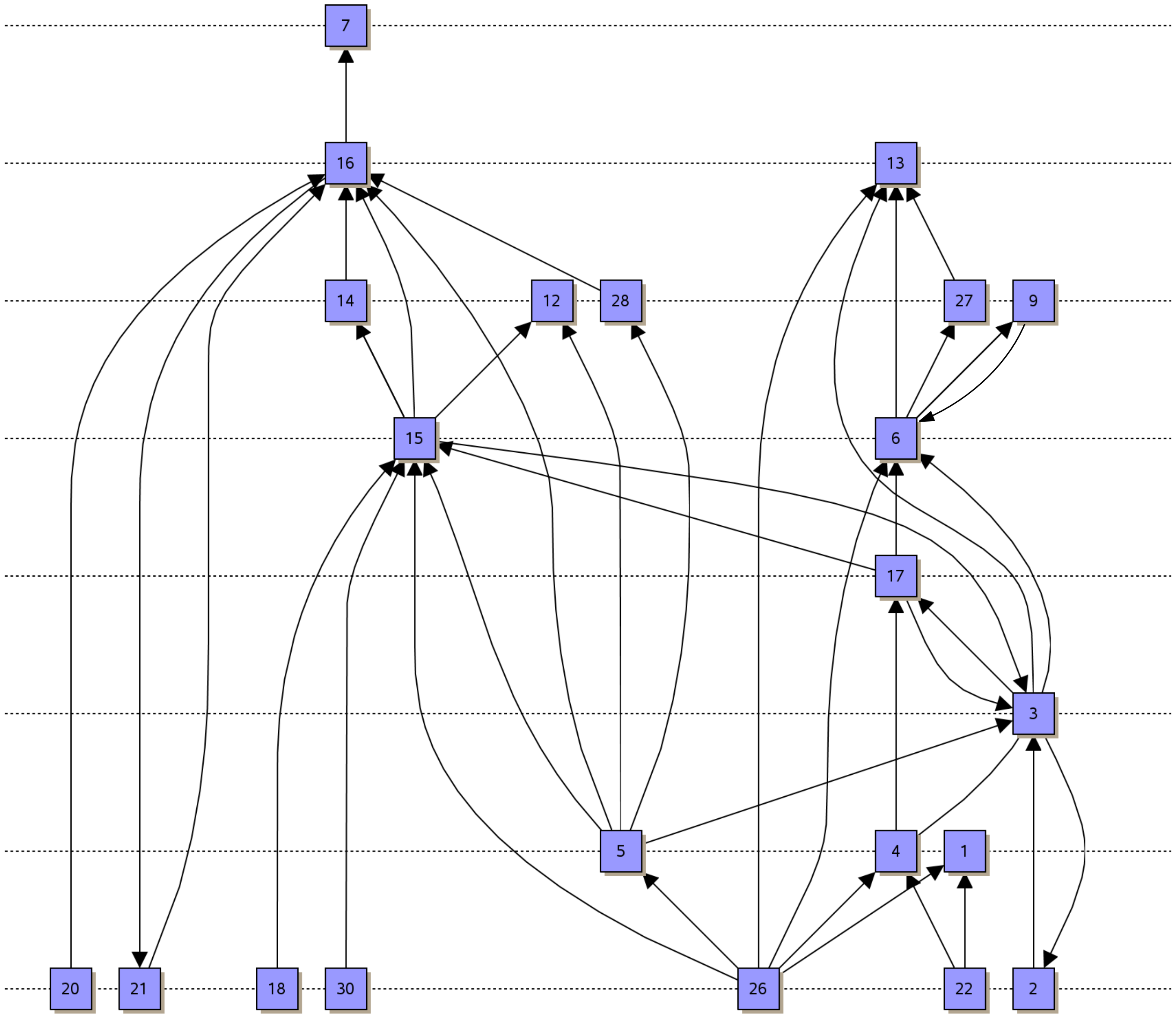
Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties.

- edges directed upwards
- vertices occur on (few) horizontal lines



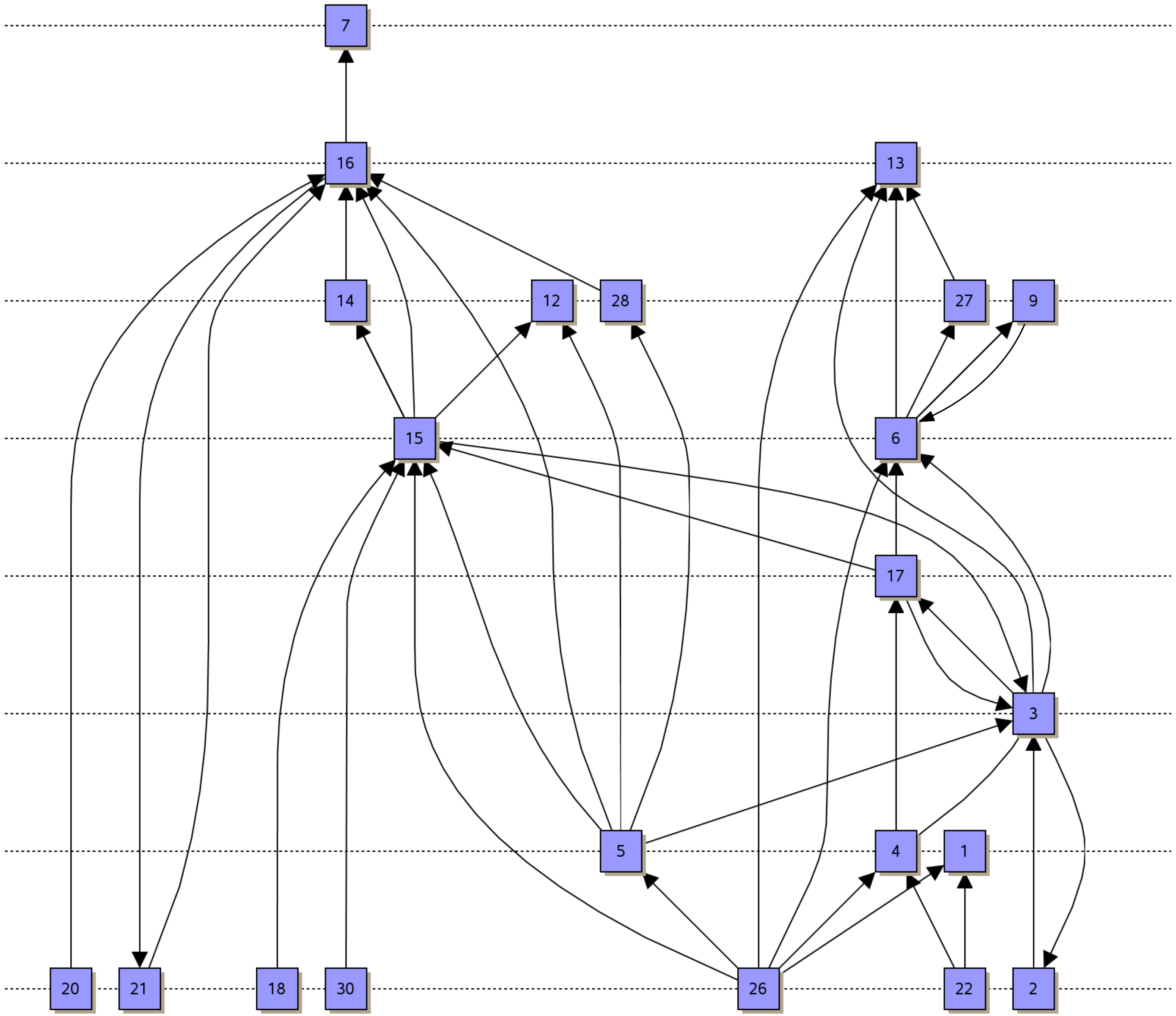
Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that “closely” reproduces the hierarchical properties of G

Desirable Properties.

- edges directed upwards
- vertices occur on (few) horizontal lines
- edge crossings minimized



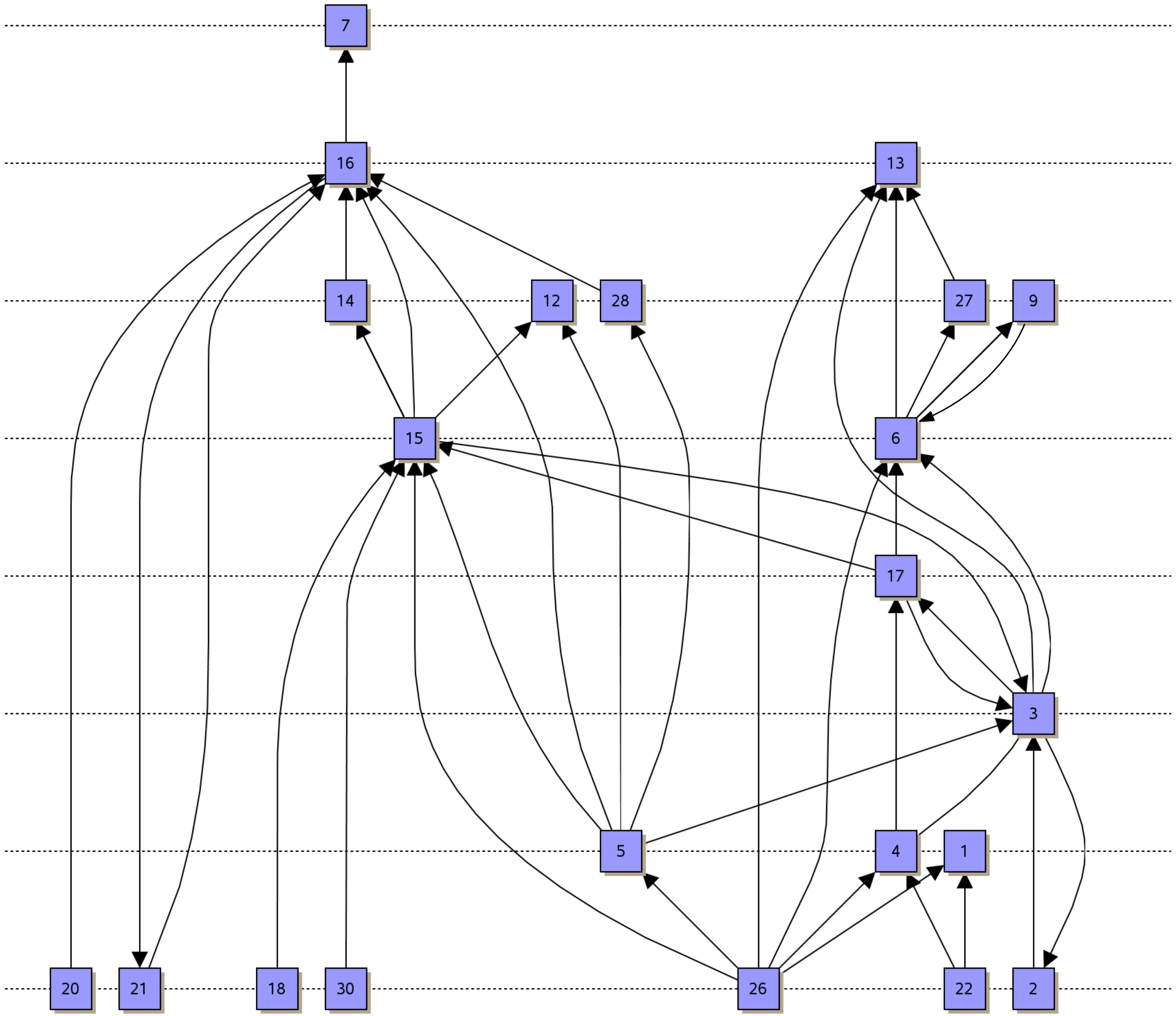
Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties.

- edges directed upwards
- vertices occur on (few) horizontal lines
- edge crossings minimized
- edges as short as possible



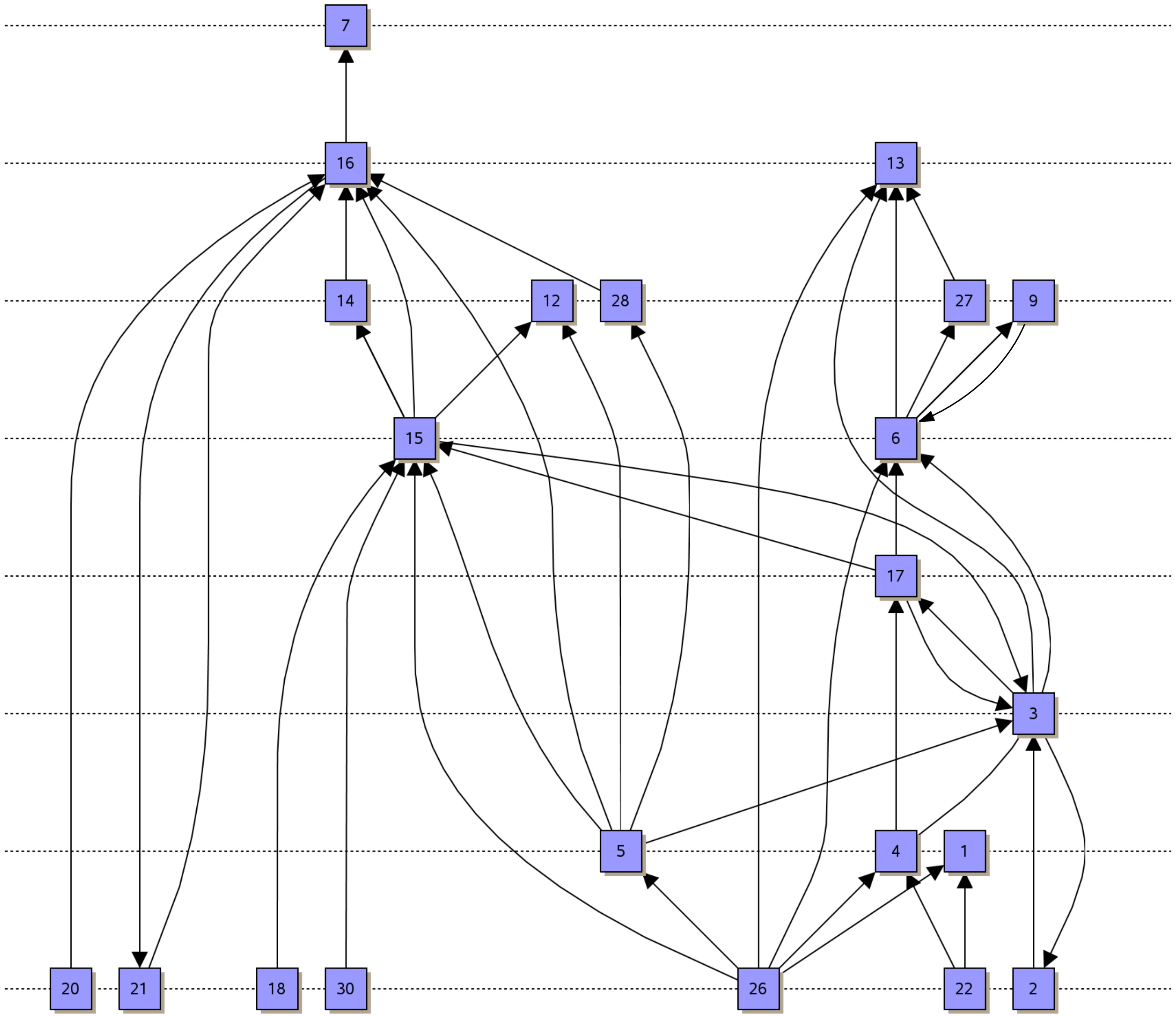
Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties.

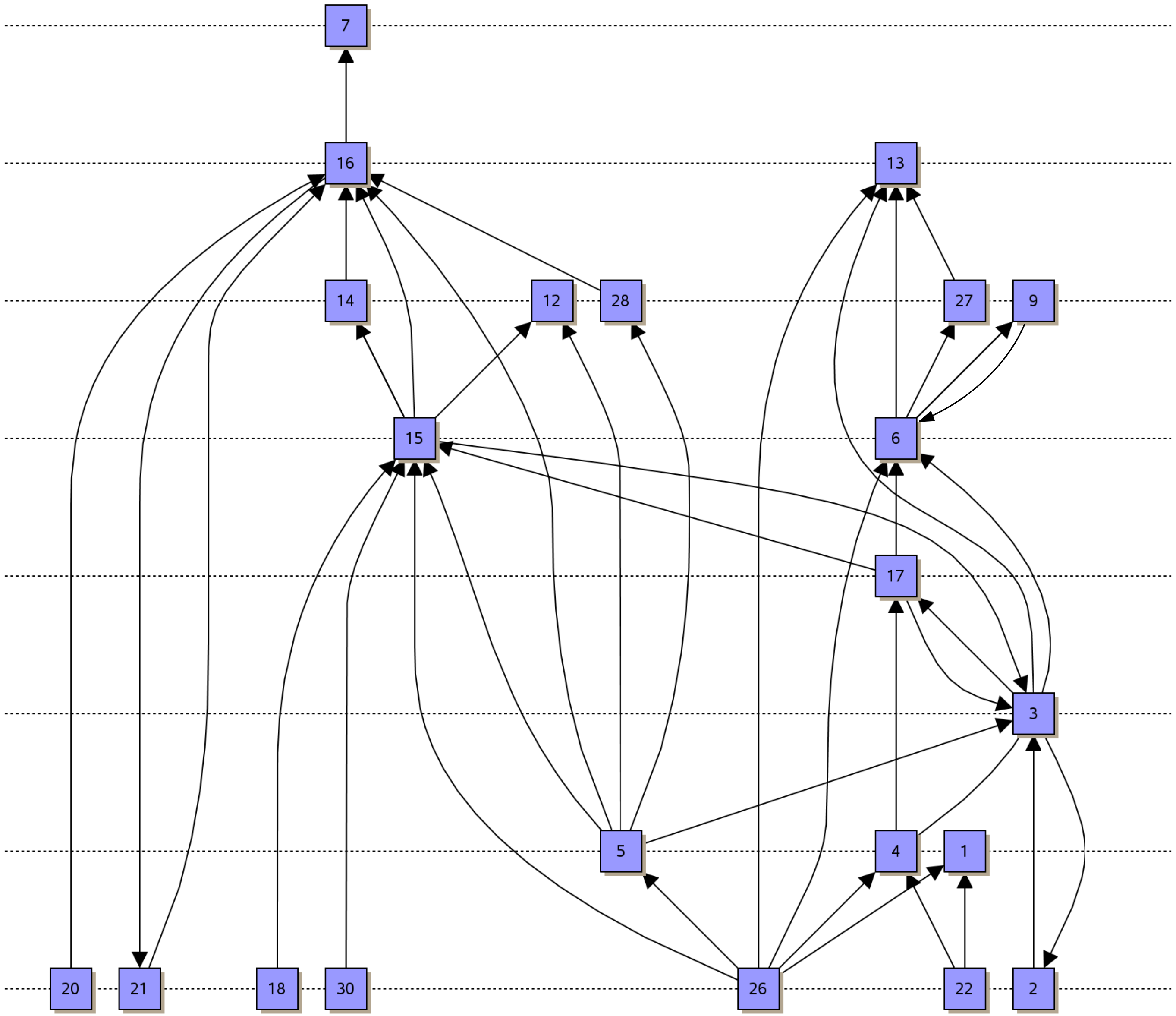
- edges directed upwards
- vertices occur on (few) horizontal lines
- edge crossings minimized
- edges as short as possible
- vertices evenly spaced



Hierarchical Drawing

Problem Statement.

- Input: digraph $G = (V, E)$
- Output: drawing of G that "closely" reproduces the hierarchical properties of G



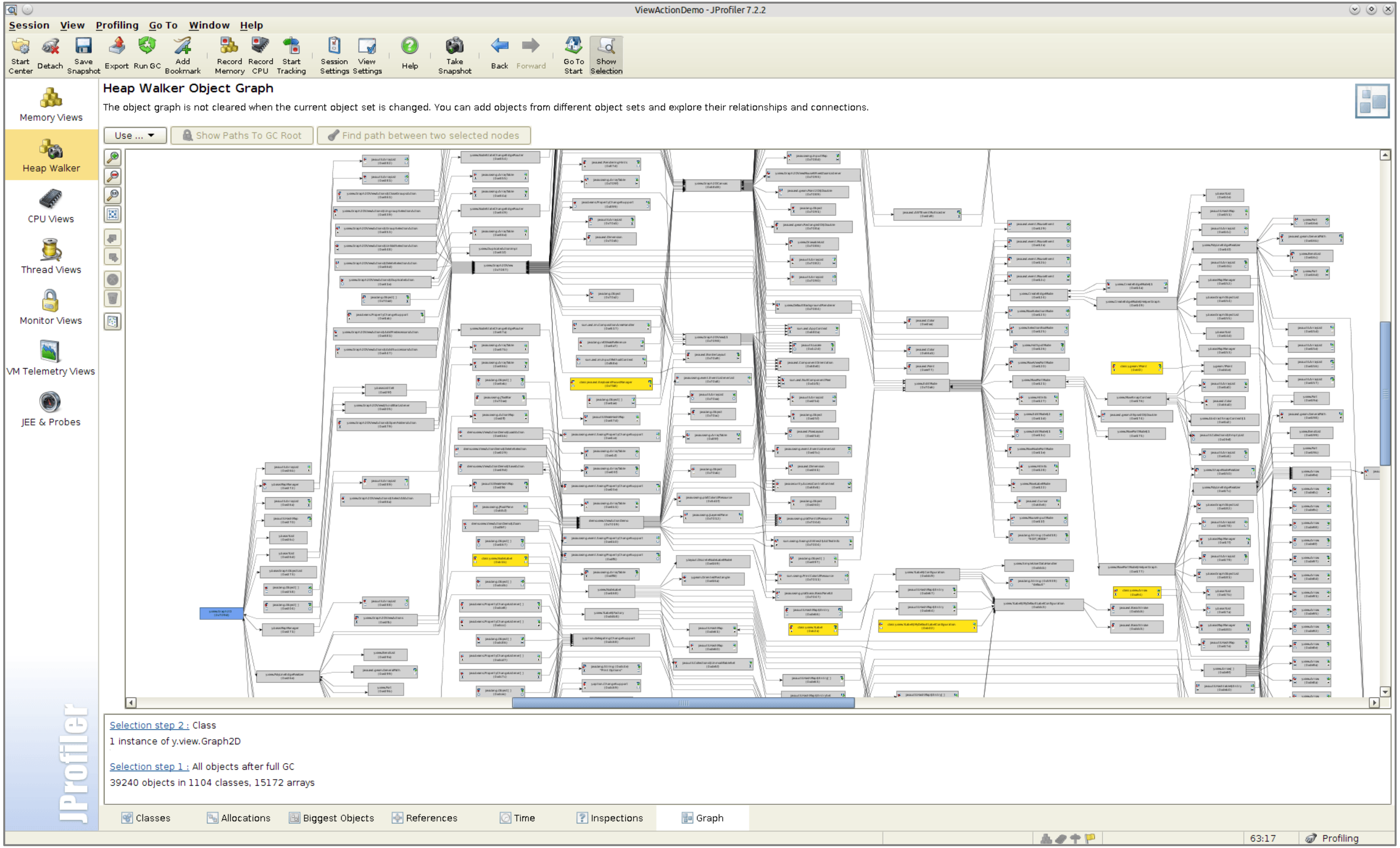
Desirable Properties.

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

Criteria can be contradictory!

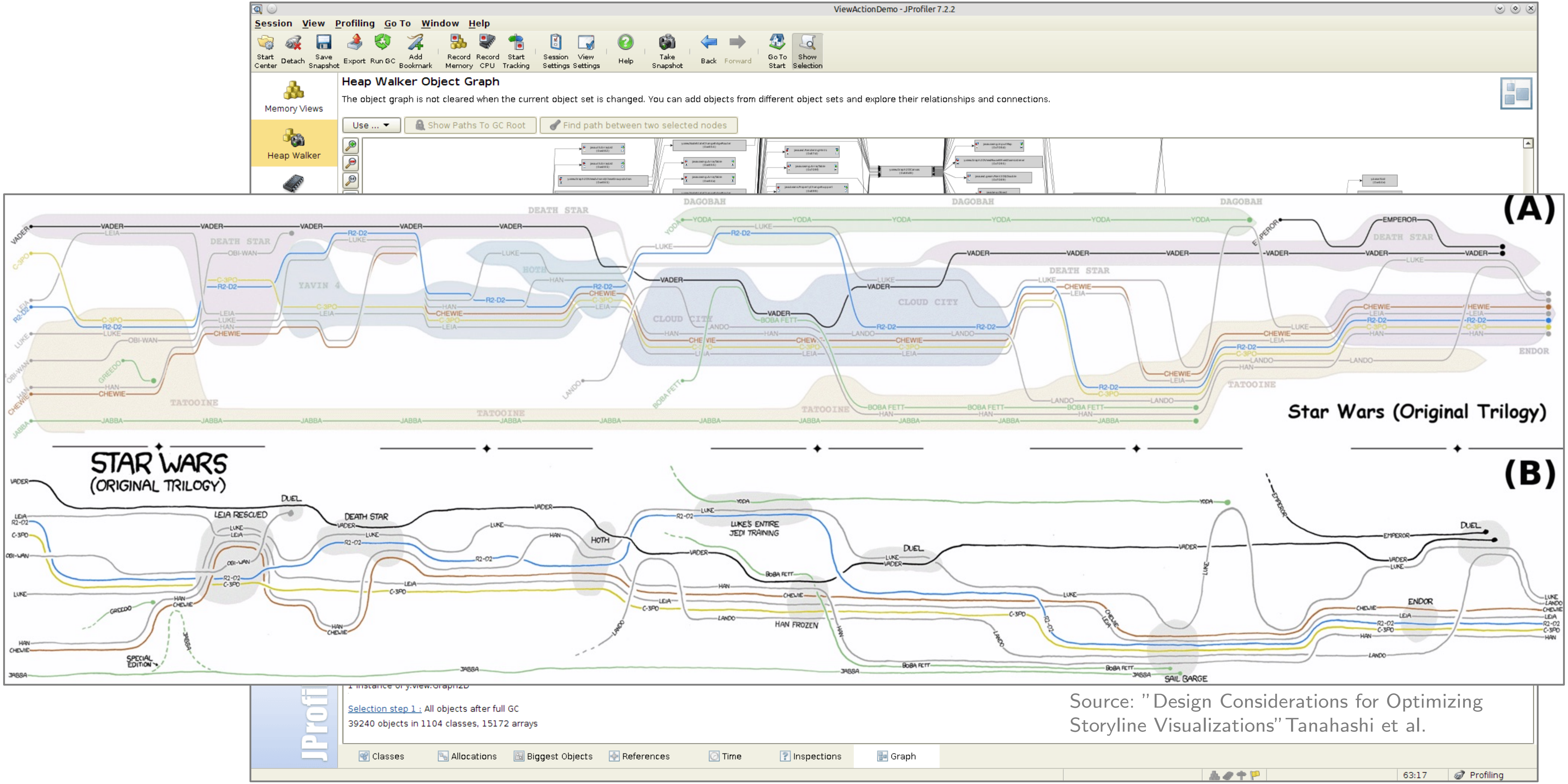
Hierarchical Drawing – Applications

yEd Gallery: Java profiler JProfiler using yFiles



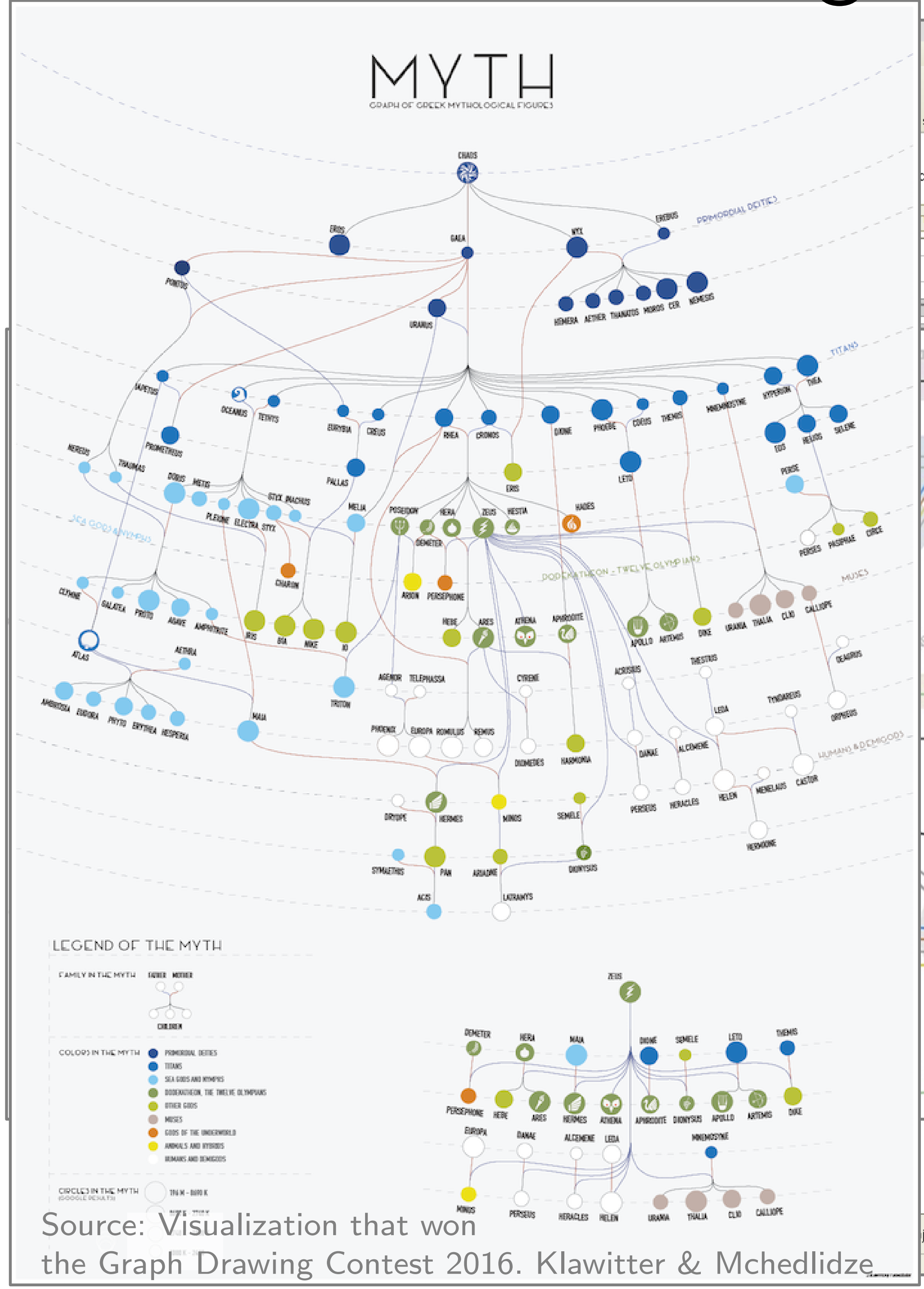
Hierarchical Drawing – Applications

yEd Gallery: Java profiler JProfiler using yFiles



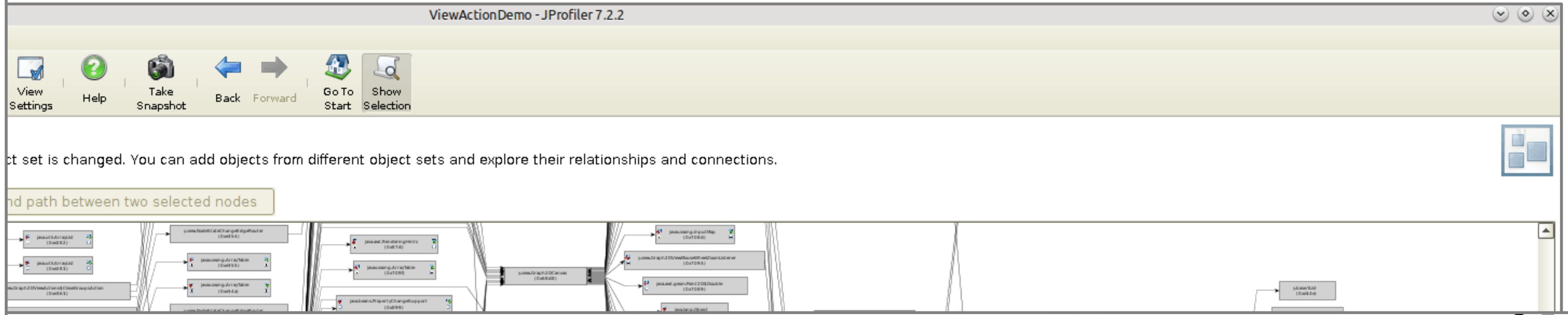
Source: "Design Considerations for Optimizing Storyline Visualizations" Tanahashi et al.

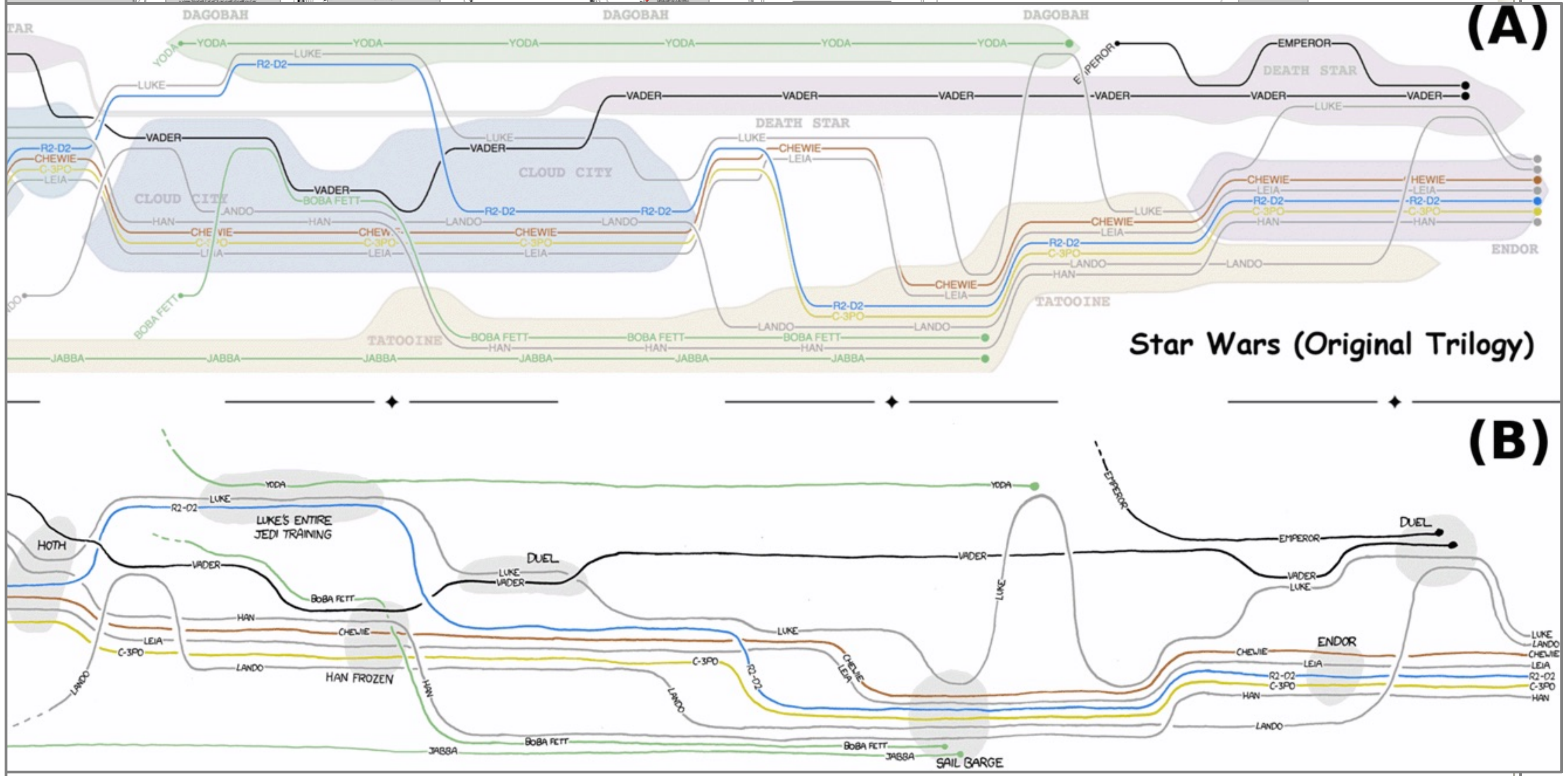
Hierarchical Drawing – Applications



Source: Visualization that won the Graph Drawing Contest 2016. Klawitter & Mchedlidze

yEd Gallery: Java profiler JProfiler using yFiles



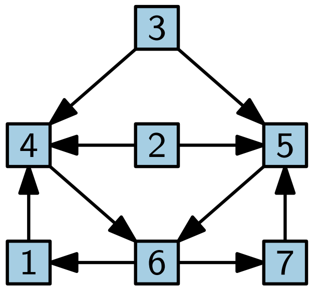


Source: "Design Considerations for Optimizing Storyline Visualizations" Tanahashi et al.

Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]

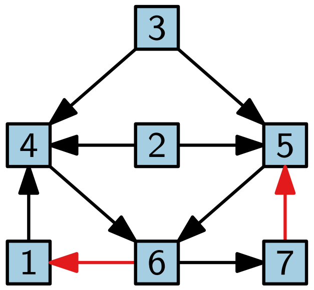
Input



Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]

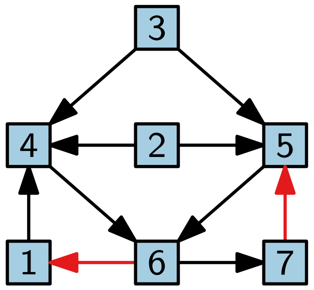
Input

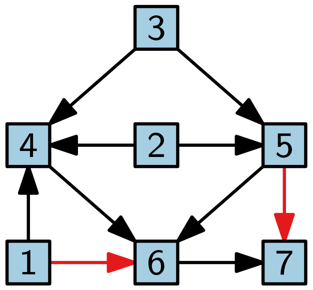


Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]

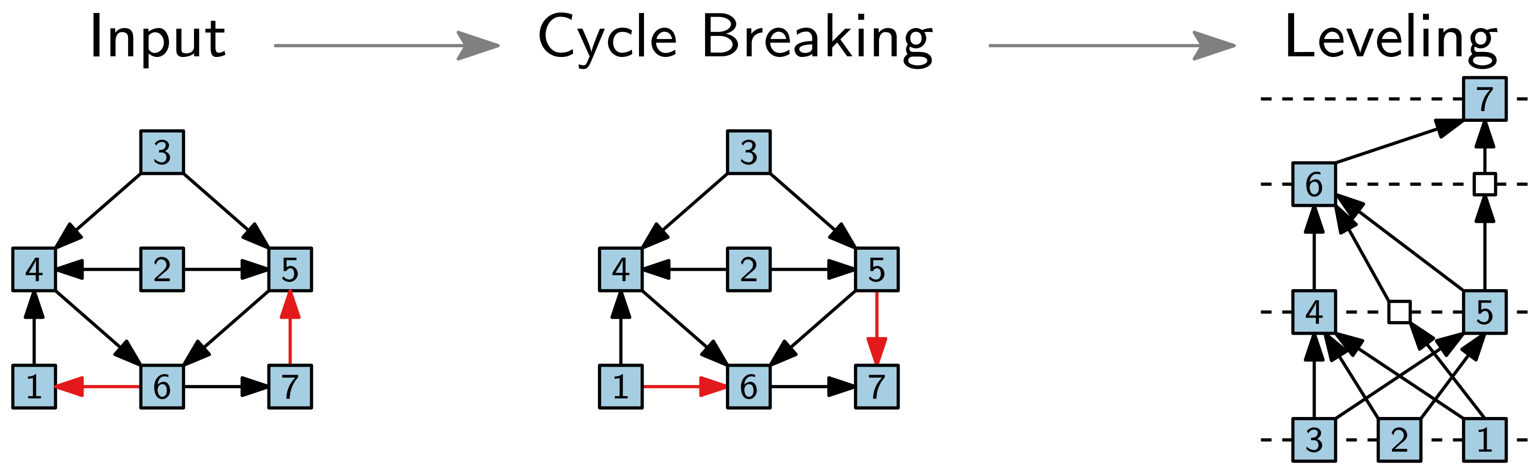
Input → Cycle Breaking





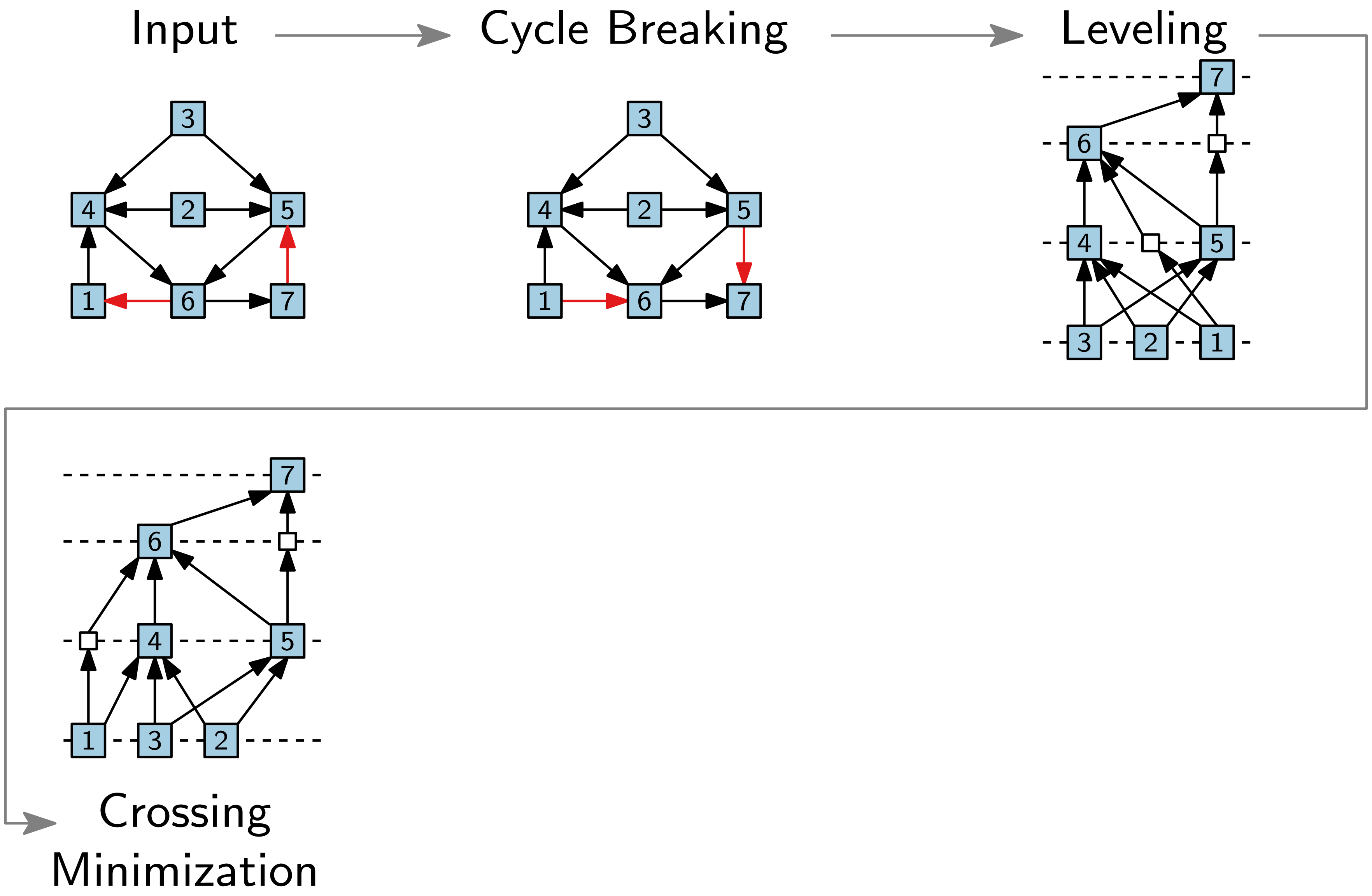
Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda ’81]



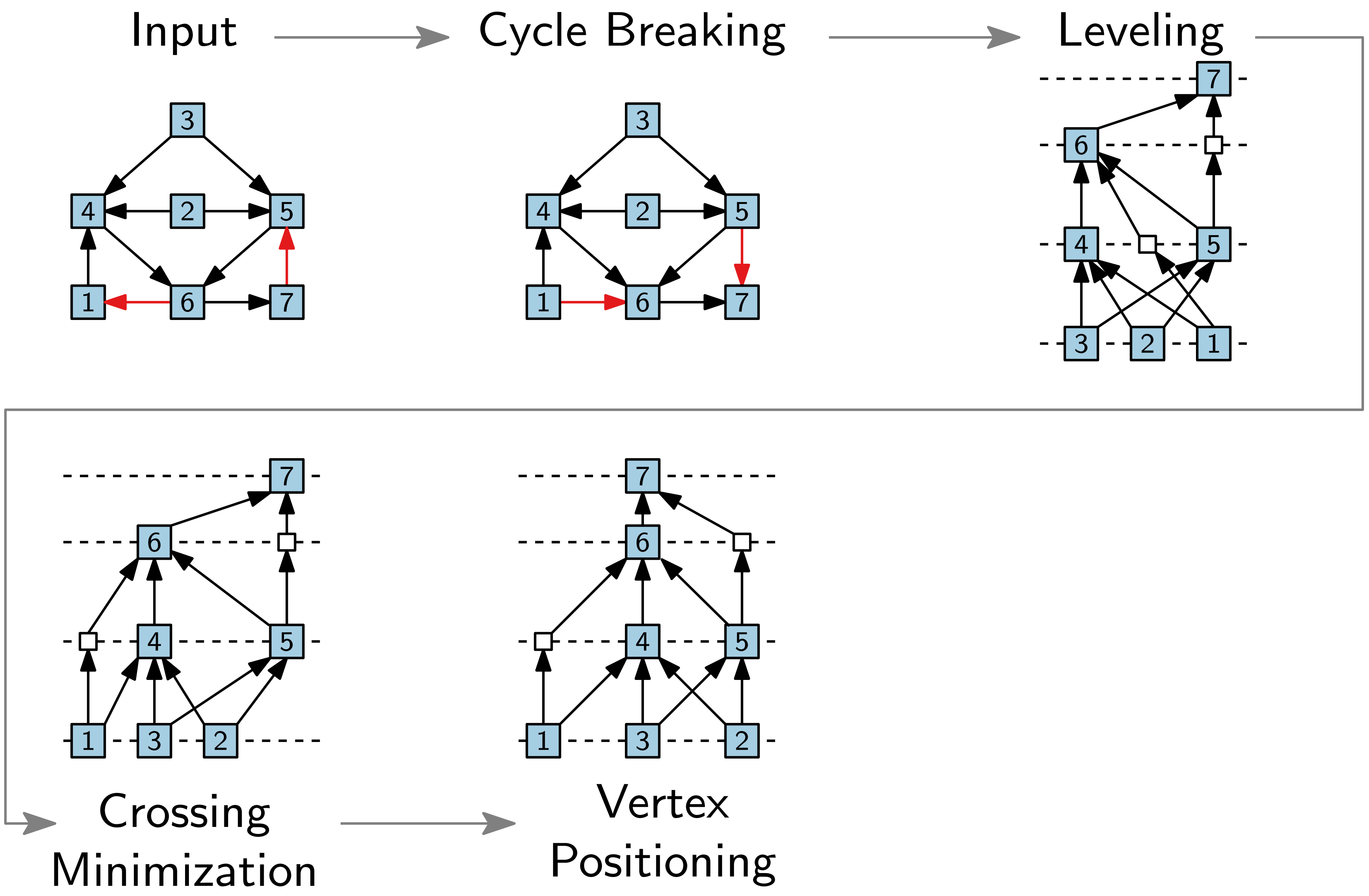
Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]



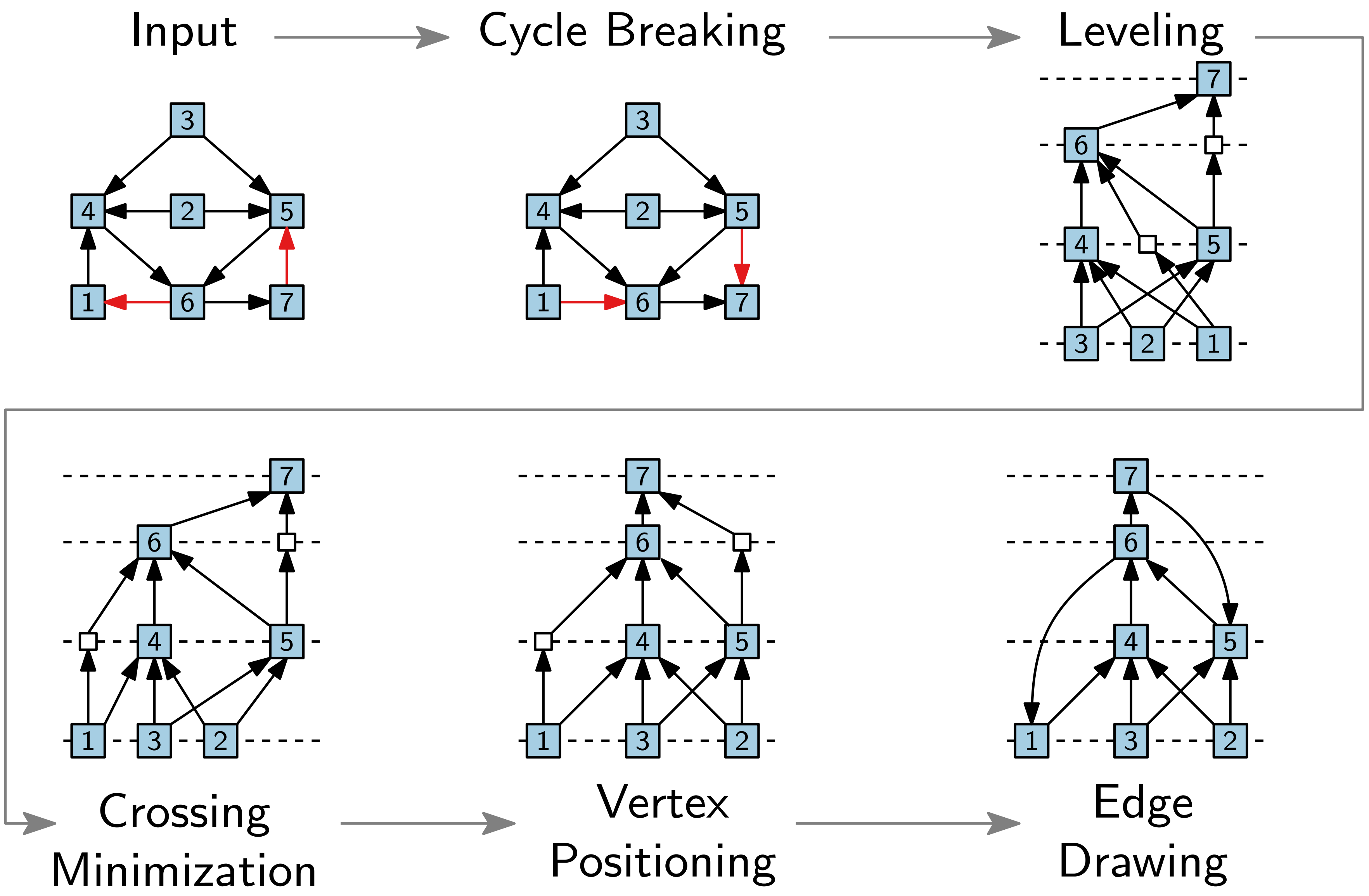
Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]

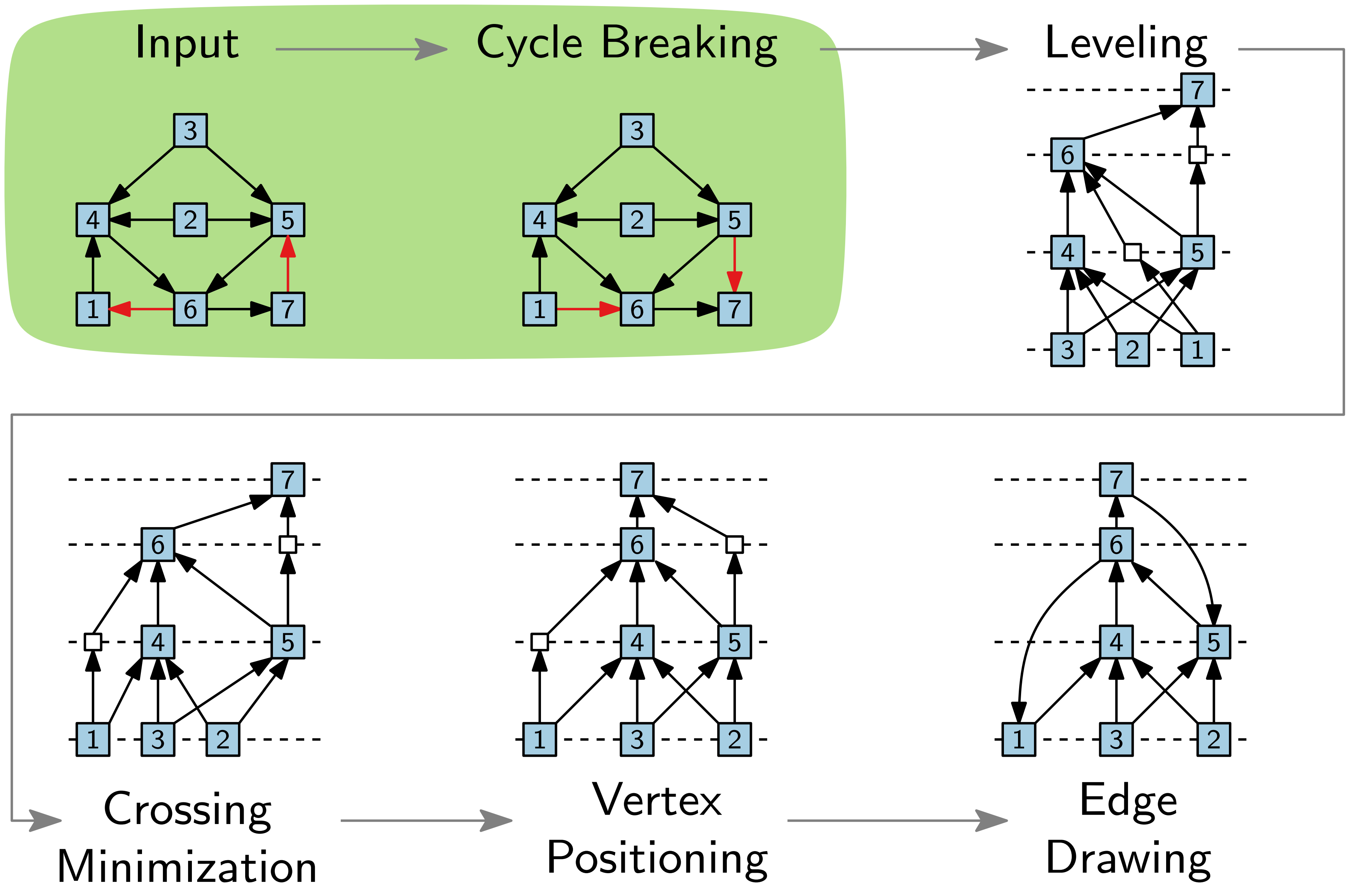


Classical Approach – Sugiyama Framework

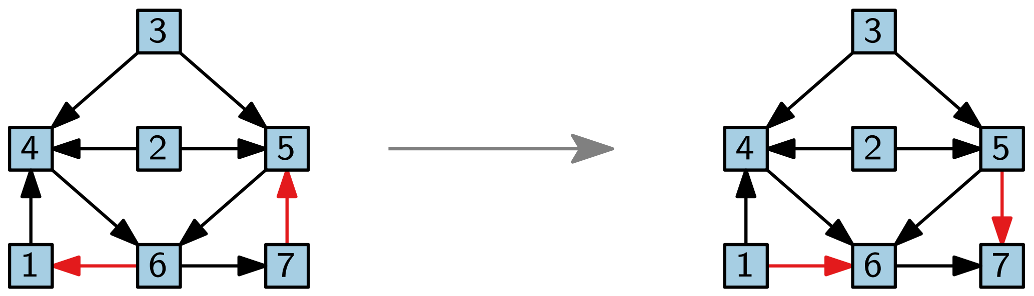
[Sugiyama, Tagawa, Toda ’81]



Step 1: Cycle breaking

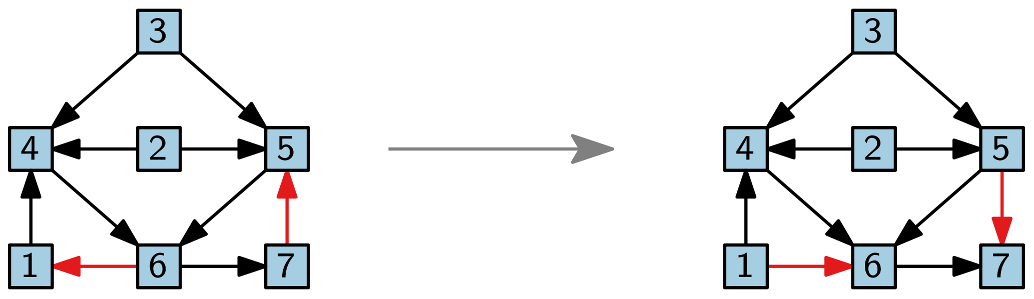


Step 1: Cycle breaking



Approach.

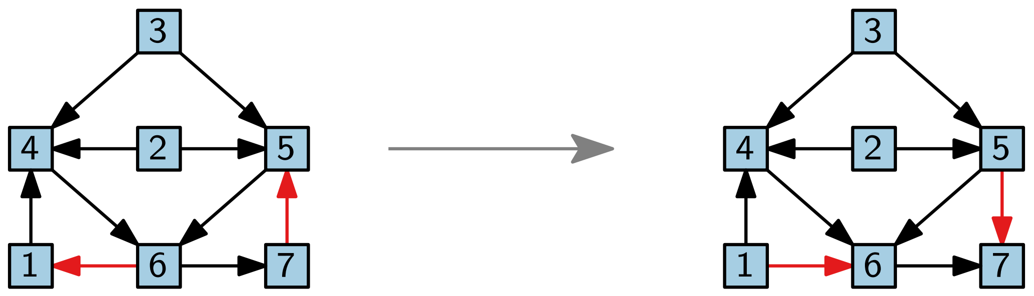
Step 1: Cycle breaking



Approach.

- Find minimum-size set $E^\star$ of edges that are not upward.

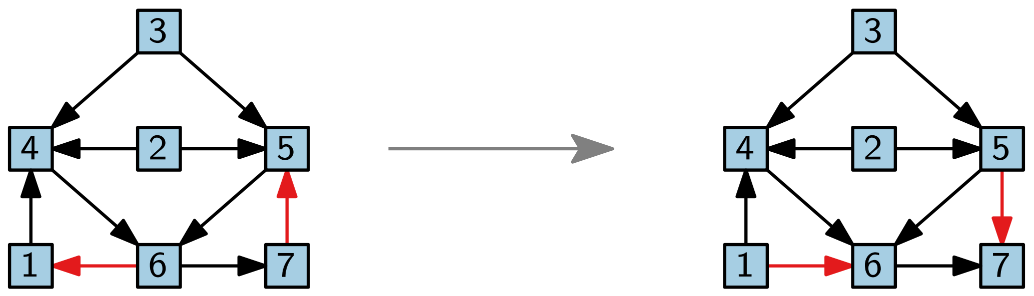
Step 1: Cycle breaking



Approach.

- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

Step 1: Cycle breaking

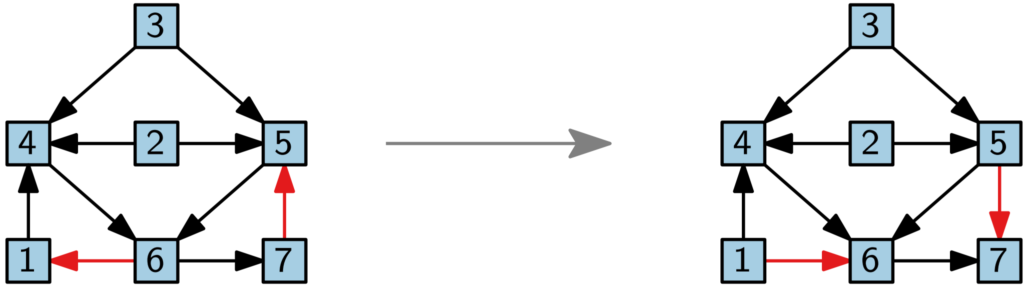


Approach.

- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

Problem MINIMUM FEEDBACK ARC SET **(FAS).**

Step 1: Cycle breaking



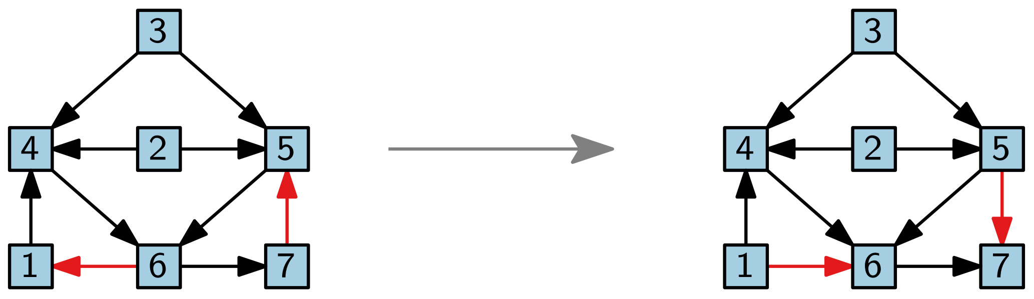
Approach.

- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

Problem MINIMUM FEEDBACK ARC SET **(FAS).**

- Input: directed graph $G = (V, E)$
- Output:

Step 1: Cycle breaking



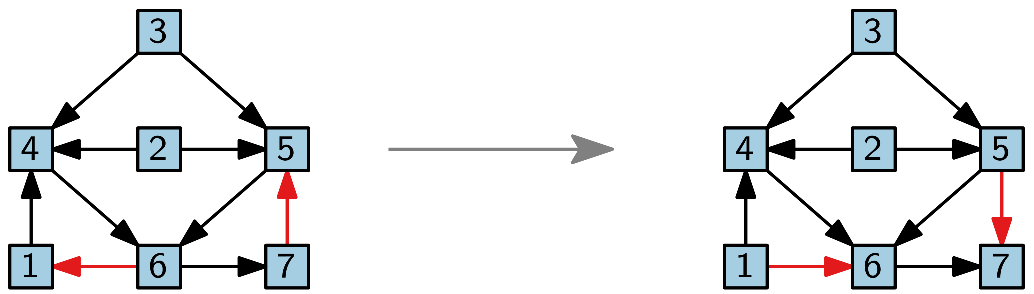
Approach.

- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

Problem MINIMUM FEEDBACK ARC SET (FAS).

- Input: directed graph $G = (V, E)$
- Output: min.-size set $E^\star \subseteq E$, such that $G^\star = (V, E \setminus E^\star)$ acyclic

Step 1: Cycle breaking



Approach.

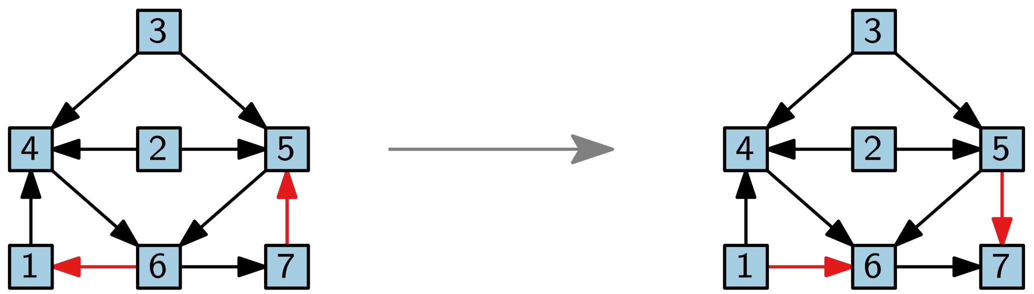
- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

Problem Minimum Feedback ~~Arc~~ Set (F~~A~~S).

- Input: directed graph $G = (V, E)$
- Output: min.-size set $E^\star \subseteq E$, such that $G^\star = (V, \cancel{E \setminus E^\star})$ acyclic

$$(E \setminus E^\star) \cup E_r^\star$$

Step 1: Cycle breaking



Approach.

- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

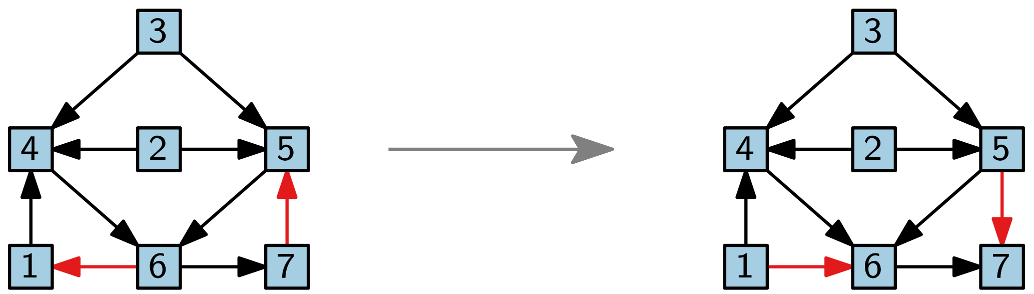
Problem Minimum Feedback ~~Arc~~ Set **(F~~A~~S).**

- Input: directed graph $G = (V, E)$
- Output: min.-size set $E^\star \subseteq E$, such that $G^\star = (V, \cancel{E \setminus E^\star})$ acyclic

edges in $E^\star$ but reversed

$$(E \setminus E^\star) \cup \boxed{E_r^\star}$$

Step 1: Cycle breaking



Approach.

- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

Problem Minimum Feedback ~~Arc~~ Set **(F~~A~~S).**

- Input: directed graph $G = (V, E)$
- Output: min.-size set $E^\star \subseteq E$, such that $G^\star = (V, \cancel{E \setminus E^\star})$ acyclic

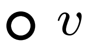
edges in $E^\star$ but reversed

$$(E \setminus E^\star) \cup E_r^\star$$

. . . NP-hard ☹

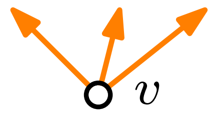
Heuristic 1

[Berger, Shor '90]



Heuristic 1

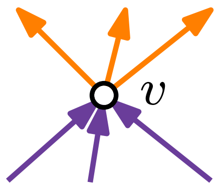
[Berger, Shor '90]



$$N^{\rightarrow}(v) \quad := \quad \{(v,u) | (v,u) \in E\}$$

Heuristic 1

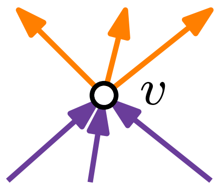
[Berger, Shor '90]



$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\}
\end{aligned}
$$

Heuristic 1

[Berger, Shor '90]

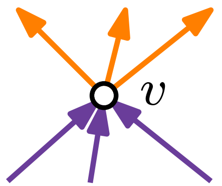


$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) \mid (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) \mid (u,v) \in E\} \\
N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v)
\end{aligned}
$$

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)



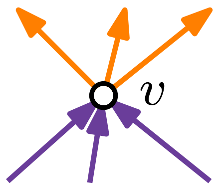
$$\begin{aligned} N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\ N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\ N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v) \end{aligned}$$

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$



$$\begin{aligned} N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\ N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\ N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v) \end{aligned}$$

return (V, E')

Heuristic 1

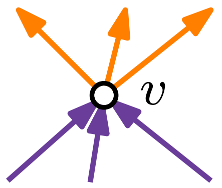
[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

return (V, E')



$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\
N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v)
\end{aligned}
$$

Heuristic 1

[Berger, Shor '90]

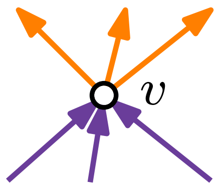
GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

return (V, E')



$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\
N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v)
\end{aligned}
$$

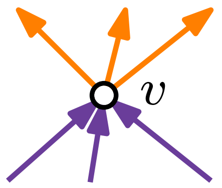
Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$
foreach $v \in V$ **do**
  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

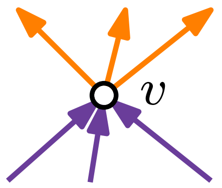
return (V, E')



$$N^{\rightarrow}(v) := \{(v, u) | (v, u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u, v) | (u, v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

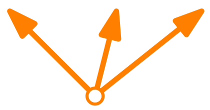
foreach $v \in V$ **do**

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

return (V, E')

$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

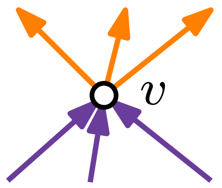
  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

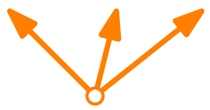
return (V, E')



$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

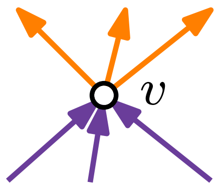
  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

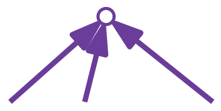
return (V, E')

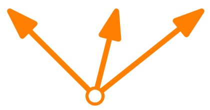


$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$





Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

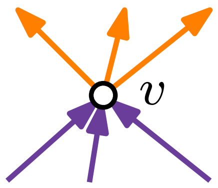
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

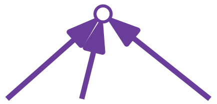
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

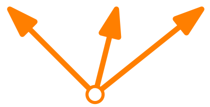
  remove v and $N(v)$ from G.

return (V, E')



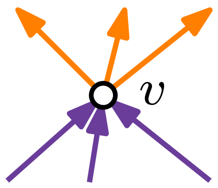
$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\
N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v)
\end{aligned}
$$





Heuristic 1

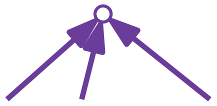
[Berger, Shor '90]



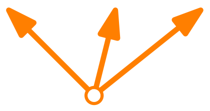
GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$
foreach $v \in V$ **do**
  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  else
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  remove v and $N(v)$ from G.
return (V, E')

$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\
N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v)
\end{aligned}
$$



- $G' = (V, E')$ is a DAG



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

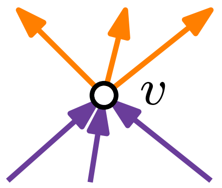
$E' \leftarrow E' \cup N^{\rightarrow}(v)$

else

$E' \leftarrow E' \cup N^{\leftarrow}(v)$

remove v and $N(v)$ from G.

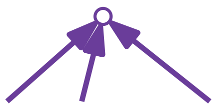
return (V, E')



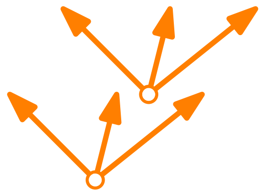
$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

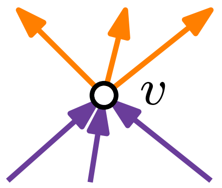


- $G' = (V, E')$ is a DAG



Heuristic 1

[Berger, Shor '90]

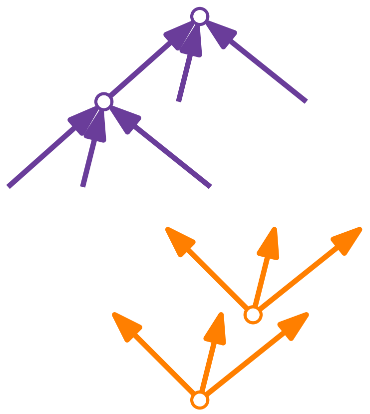


GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$
foreach $v \in V$ **do**
  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  else
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  remove v and $N(v)$ from G.
return (V, E')

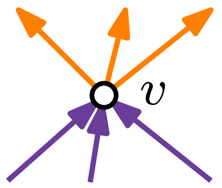
$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

- $G' = (V, E')$ is a DAG



Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

  remove v and $N(v)$ from G.

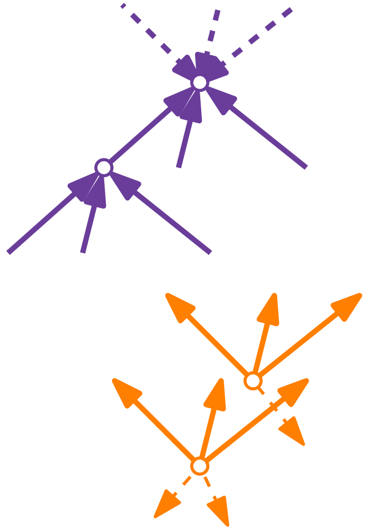
return (V, E')

$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

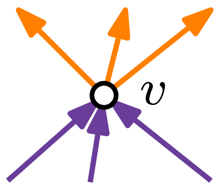
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

- $G' = (V, E')$ is a DAG



Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

$E' \leftarrow E' \cup N^{\rightarrow}(v)$

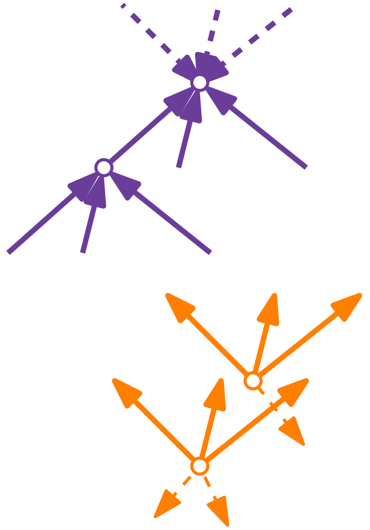
else

$E' \leftarrow E' \cup N^{\leftarrow}(v)$

remove v and $N(v)$ from G.

return (V, E')

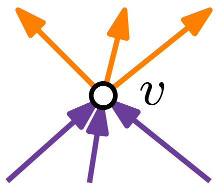
$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$



- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set

Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

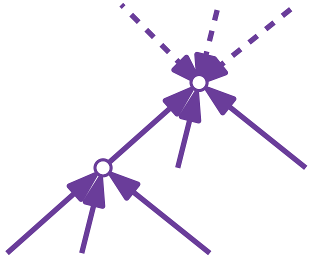
  remove v and $N(v)$ from G.

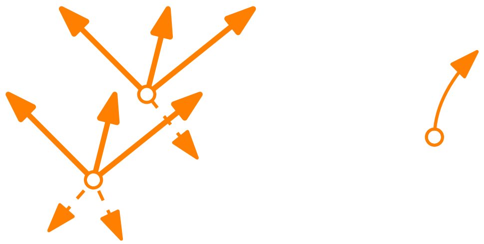
return (V, E')

$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

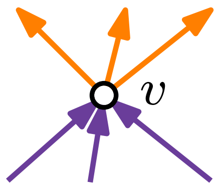




- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set

Heuristic 1

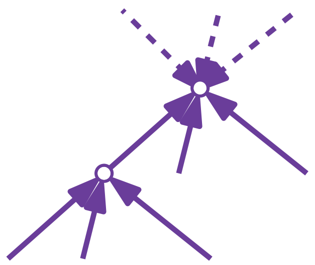
[Berger, Shor '90]



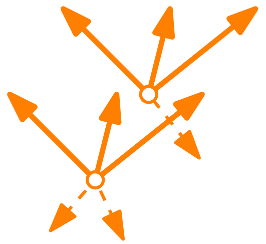
GreedyMakeAcyclic(Digraph $G = (V, E)$)

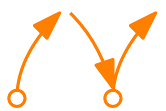
$E' \leftarrow \emptyset$
foreach $v \in V$ **do**
  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  else
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  remove v and $N(v)$ from G.
return (V, E')

$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\
N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v)
\end{aligned}
$$



- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set





Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

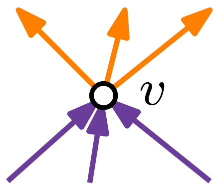
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

  remove v and $N(v)$ from G.

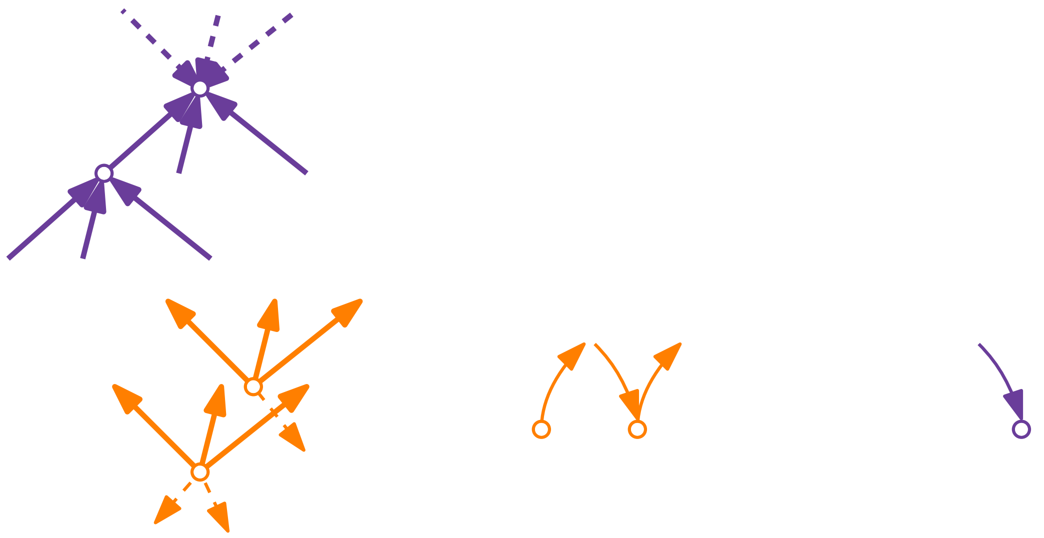
return (V, E')



$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

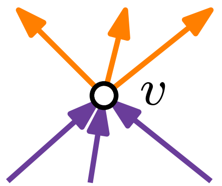
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$



- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set

Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

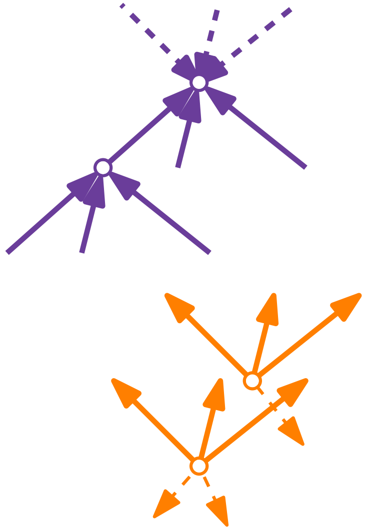
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

  remove v and $N(v)$ from G.

return (V, E')

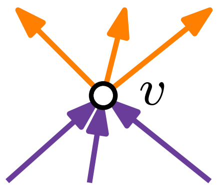
$$\begin{aligned} N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\ N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\ N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v) \end{aligned}$$

- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set



Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

  remove v and $N(v)$ from G.

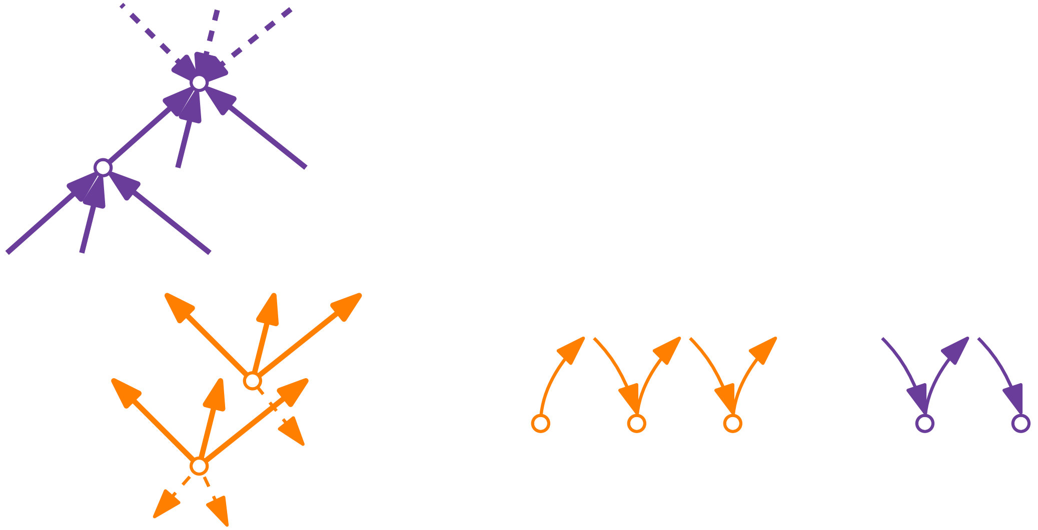
return (V, E')

$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

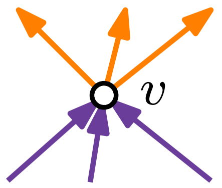
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set



Heuristic 1

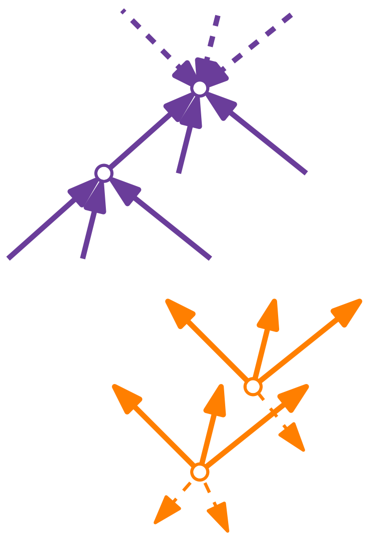
[Berger, Shor '90]



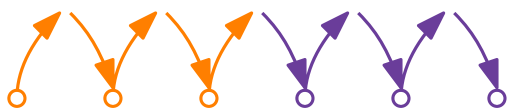
GreedyMakeAcyclic(Digraph $G = (V, E)$)

  $E' \leftarrow \emptyset$
  foreach $v \in V$ **do**
    if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**
      $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    else
      $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N(v)$ from G.
  return (V, E')

$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

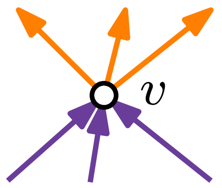


- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set



Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

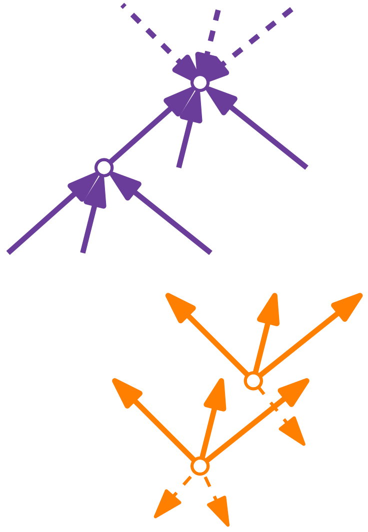
  remove v and $N(v)$ from G.

return (V, E')

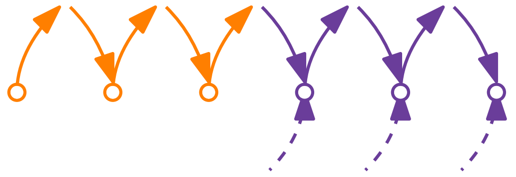
$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$

$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$

$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

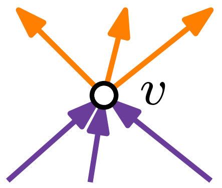


- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set



Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

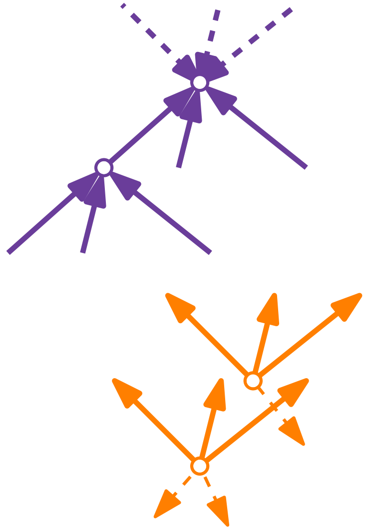
  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

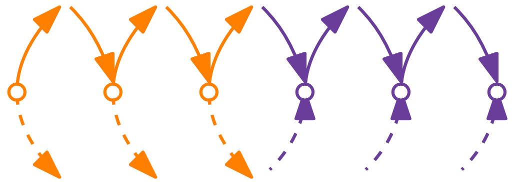
  remove v and $N(v)$ from G.

return (V, E')

$$
\begin{aligned}
N^{\rightarrow}(v) &:= \{(v,u) | (v,u) \in E\} \\
N^{\leftarrow}(v) &:= \{(u,v) | (u,v) \in E\} \\
N(v) &:= N^{\rightarrow}(v) \cup N^{\leftarrow}(v)
\end{aligned}
$$

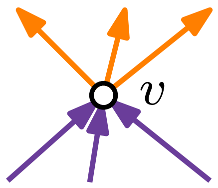


- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set



Heuristic 1

[Berger, Shor '90]



GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

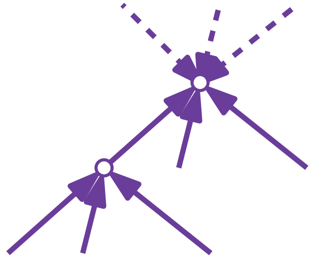
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

  remove v and $N(v)$ from G.

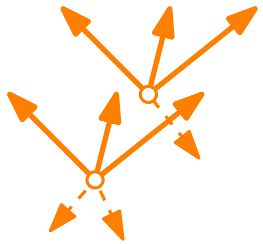
return (V, E')

$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

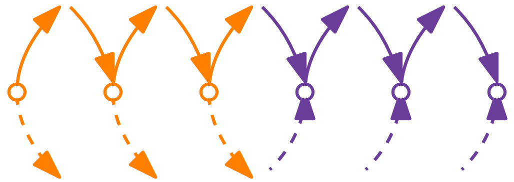
- Time:



- $G' = (V, E')$ is a DAG



- $E \setminus E'$ is a feedback set

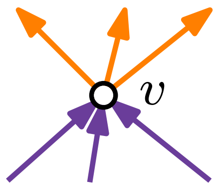


Heuristic 1

[Berger, Shor '90]

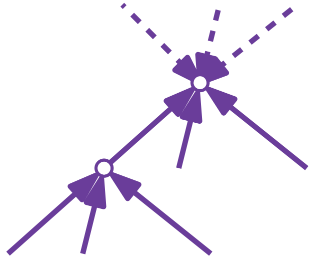
GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$
foreach $v \in V$ **do**
  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  else
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  remove v and $N(v)$ from G.
return (V, E')

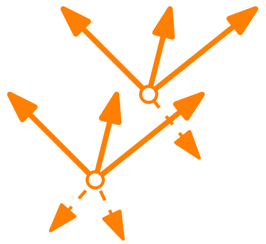


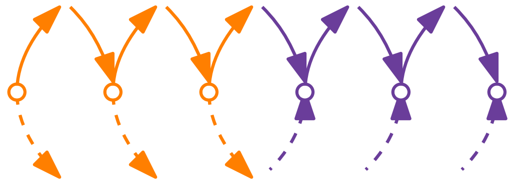
$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

- Time: $\mathcal{O}(|V| + |E|)$



- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set





Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

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

  if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**

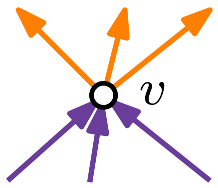
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

  else

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

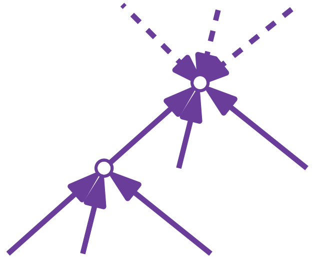
  remove v and $N(v)$ from G.

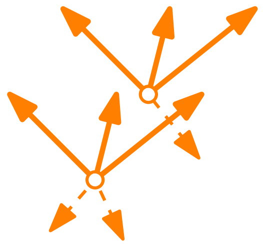
return (V, E')



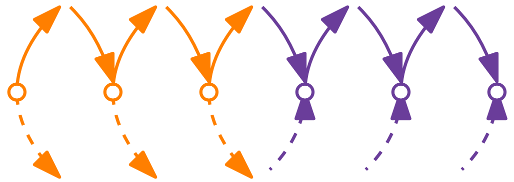
$$N^{\rightarrow}(v) := \{(v,u) | (v,u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u,v) | (u,v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

- Time: $\mathcal{O}(|V| + |E|)$
- Quality guarantee: $|E'| \geq$



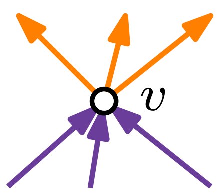


- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set



Heuristic 1

[Berger, Shor '90]

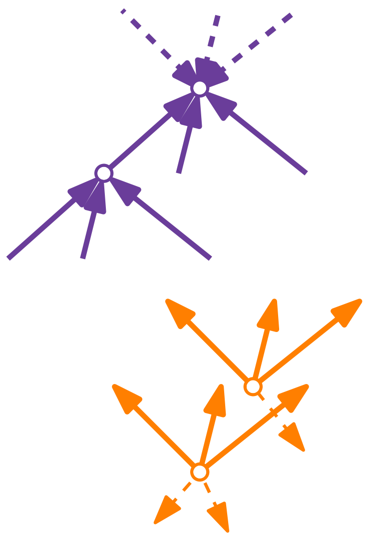


GreedyMakeAcyclic(Digraph $G = (V, E)$)

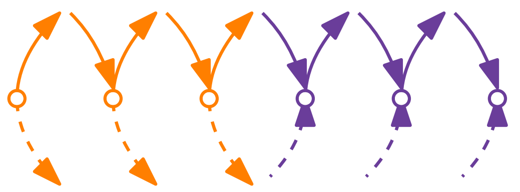
$E' \leftarrow \emptyset$
foreach $v \in V$ **do**
    if $|N^{\rightarrow}(v)| \geq |N^{\leftarrow}(v)|$ **then**
        $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    else
        $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N(v)$ from G.
return (V, E')

$$N^{\rightarrow}(v) := \{(v, u) | (v, u) \in E\}$$
$$N^{\leftarrow}(v) := \{(u, v) | (u, v) \in E\}$$
$$N(v) := N^{\rightarrow}(v) \cup N^{\leftarrow}(v)$$

- Time: $\mathcal{O}(|V| + |E|)$
- Quality guarantee: $|E'| \geq |E|/2$

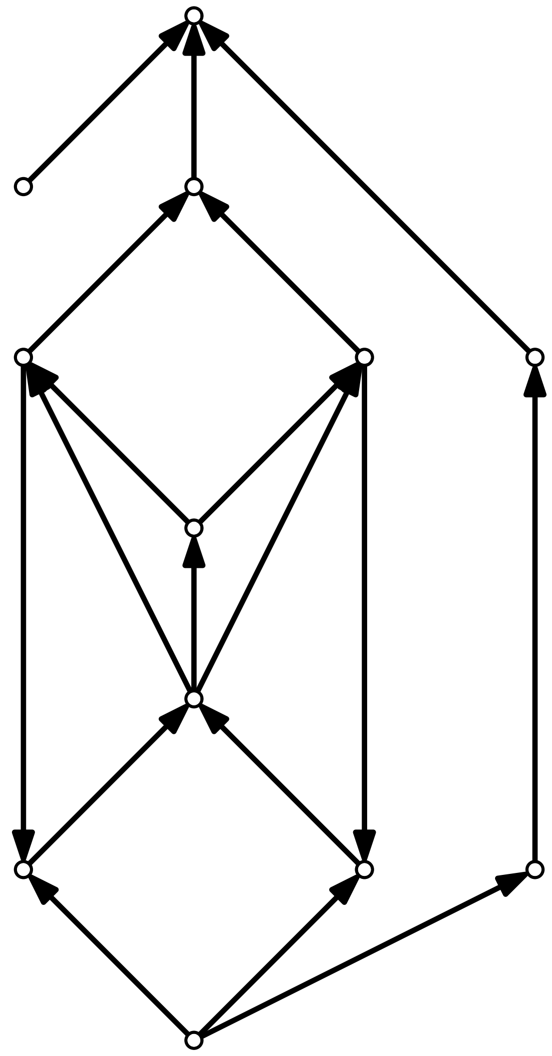


- $G' = (V, E')$ is a DAG
- $E \setminus E'$ is a feedback set



Heuristic 2

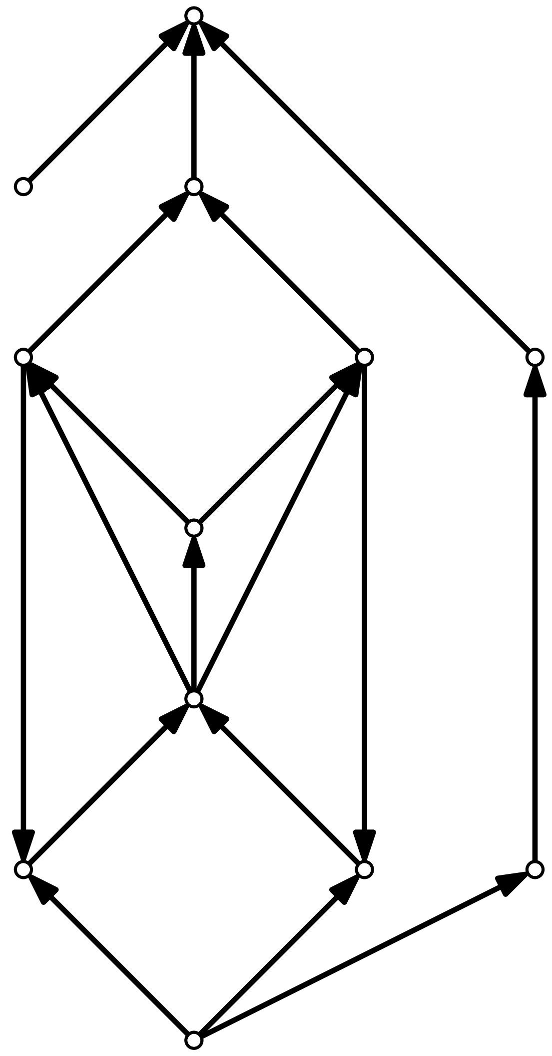
[Eades, Lin, Smyth '93]



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

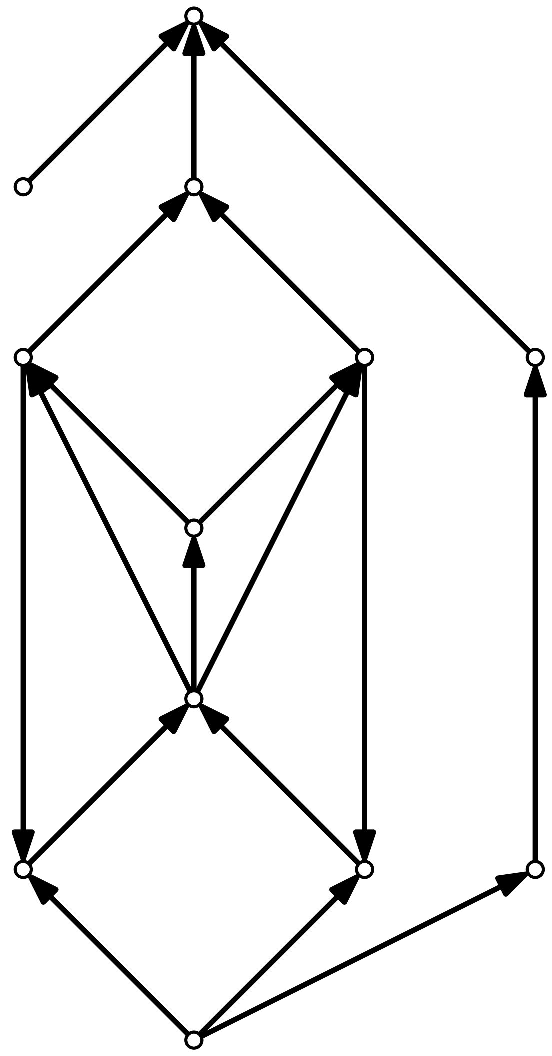


Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**



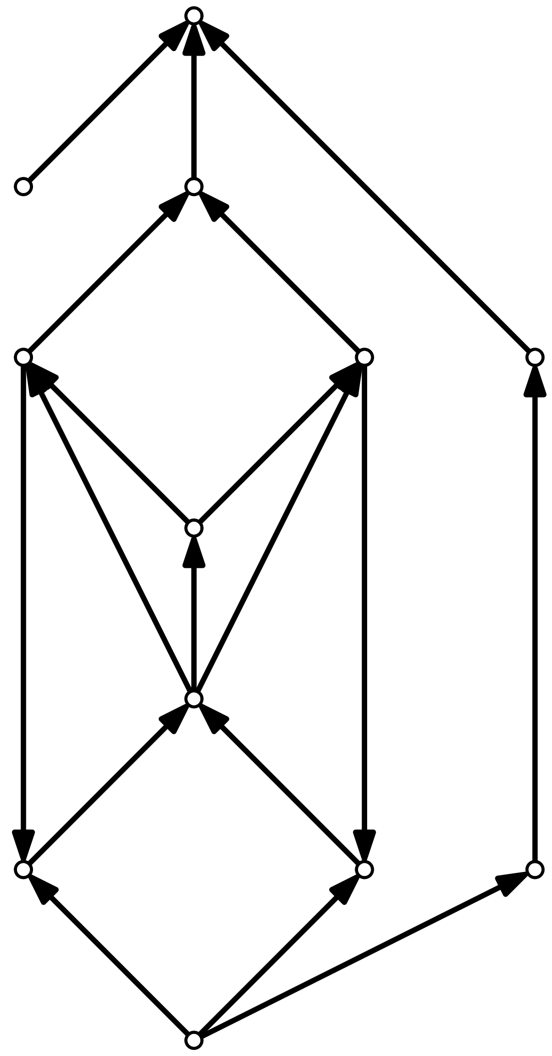
Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**



Heuristic 2

[Eades, Lin, Smyth '93]

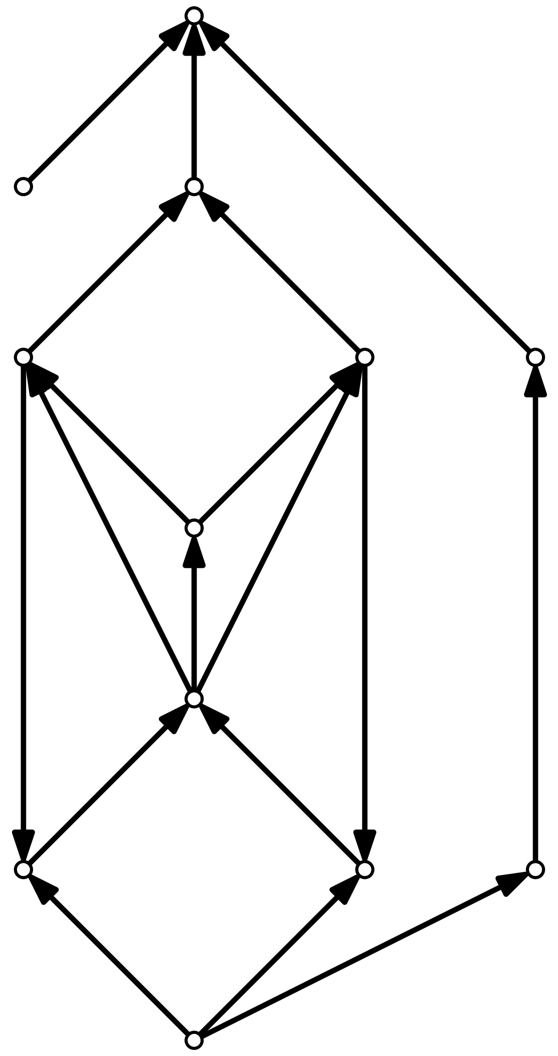
$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

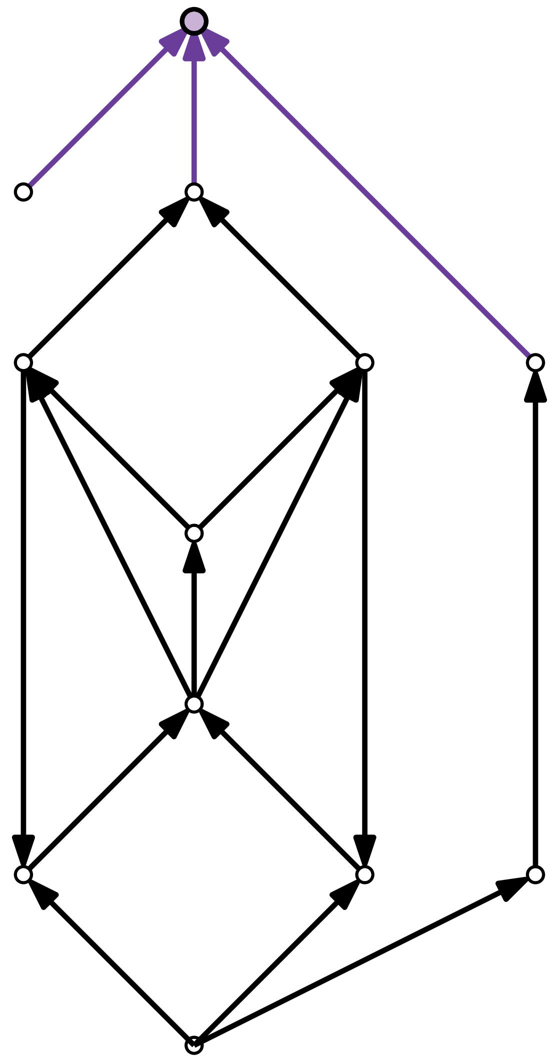
$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

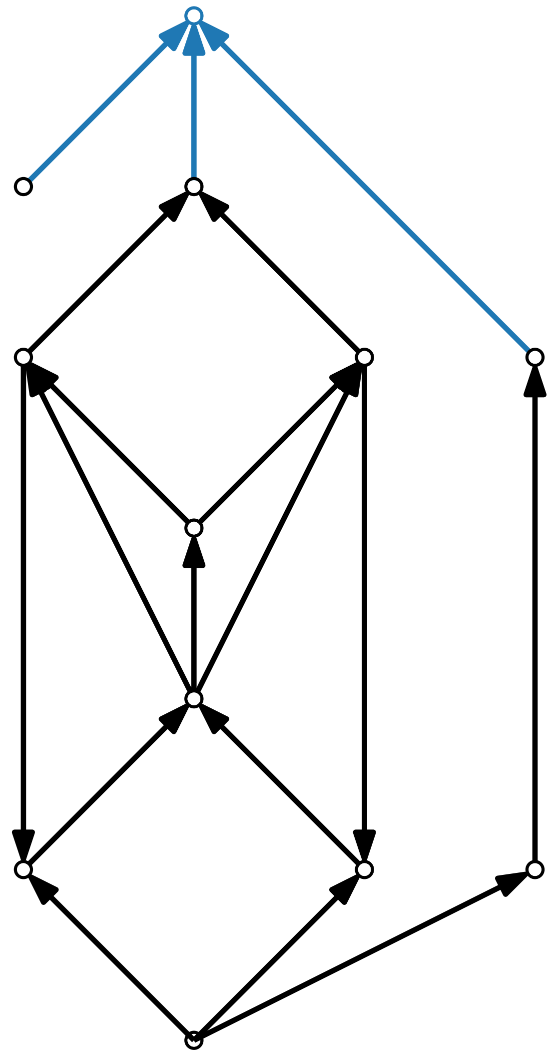
$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

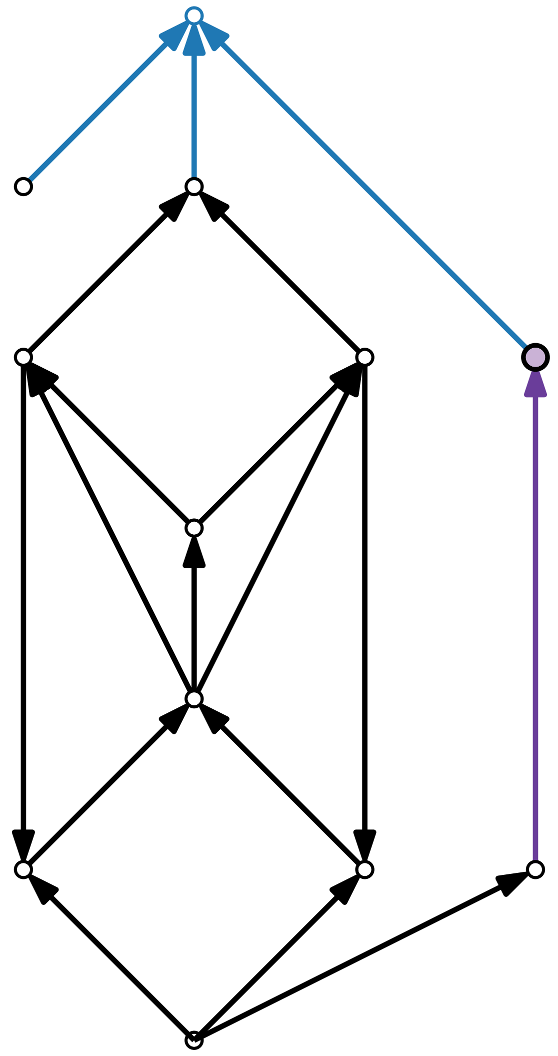
    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

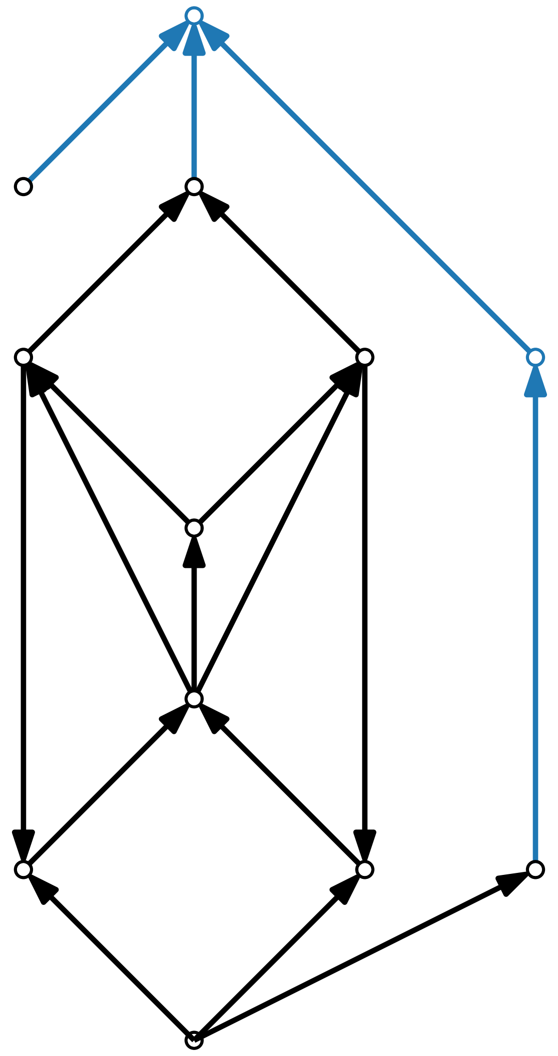
$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

    while in V exists a sink v **do**

        $E' \leftarrow E' \cup N^{\leftarrow}(v)$

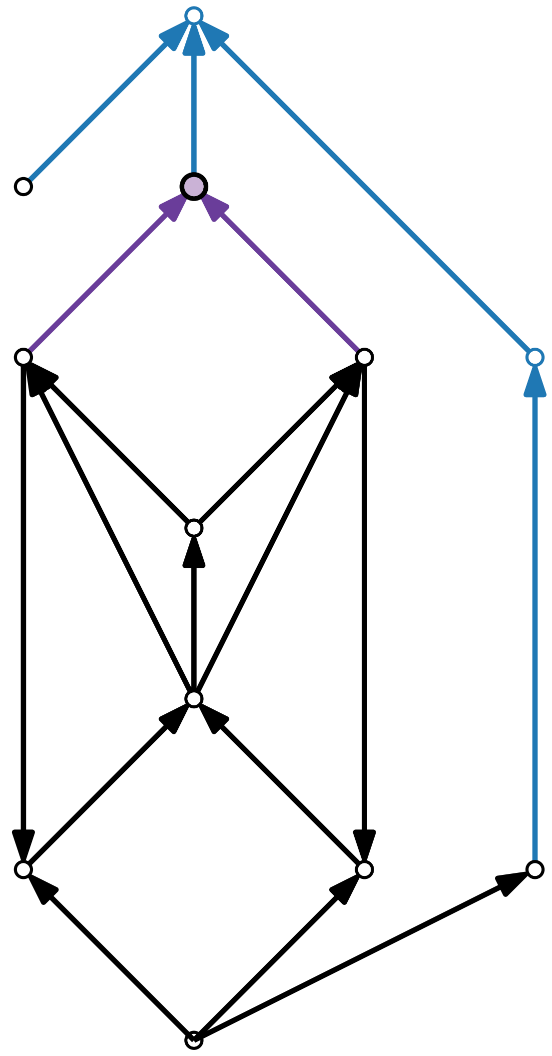
        remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

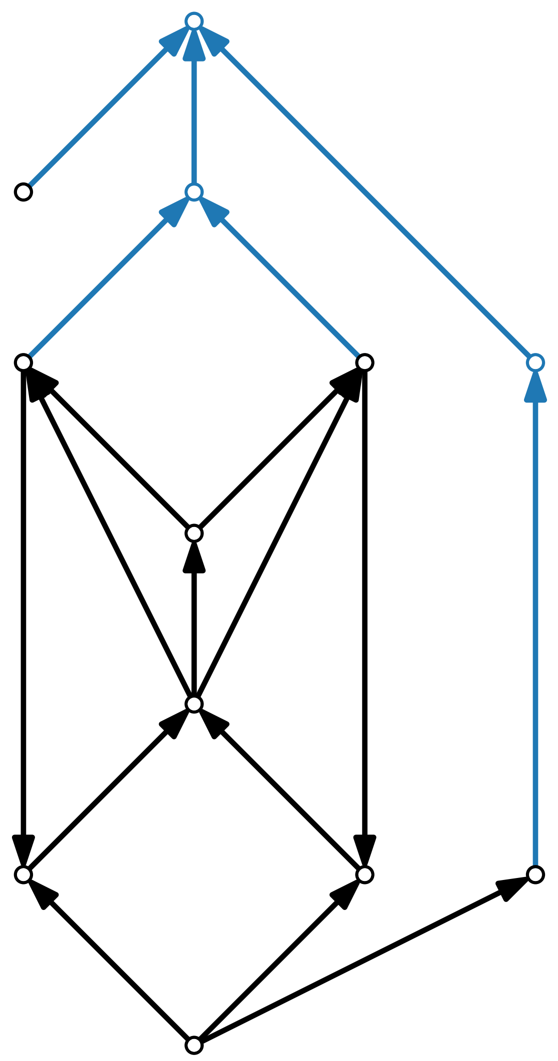
$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

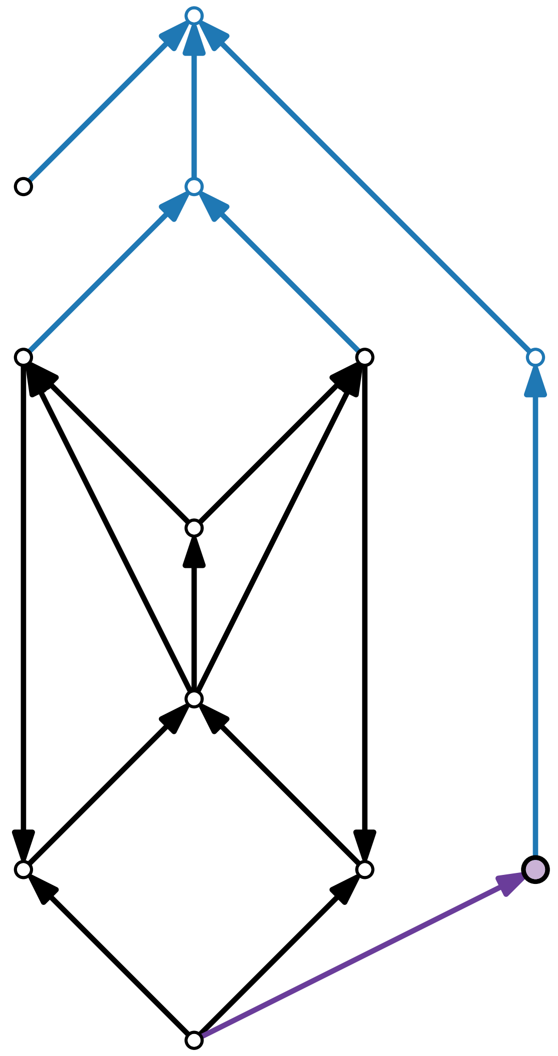
    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

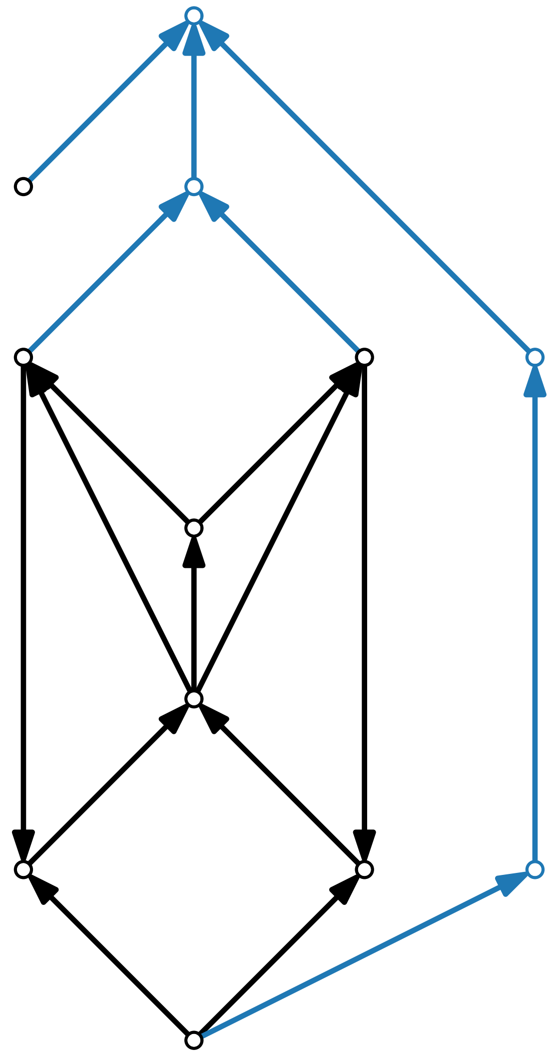
$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

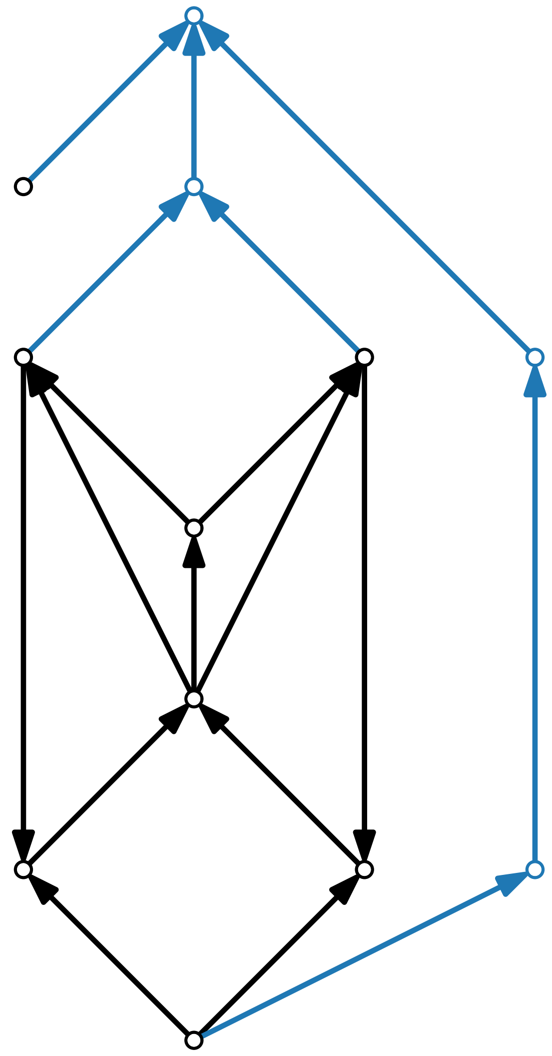
while $V \neq \emptyset$ **do**

    while in V exists a sink v **do**

        $E' \leftarrow E' \cup N^{\leftarrow}(v)$

        remove v and $N^{\leftarrow}(v)$

    Remove all isolated vertices from V



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

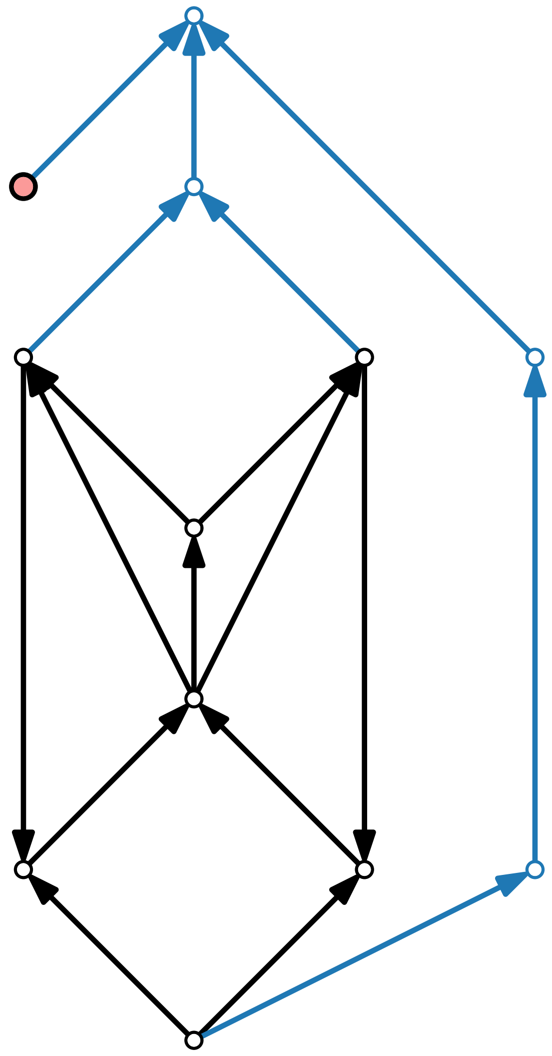
while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

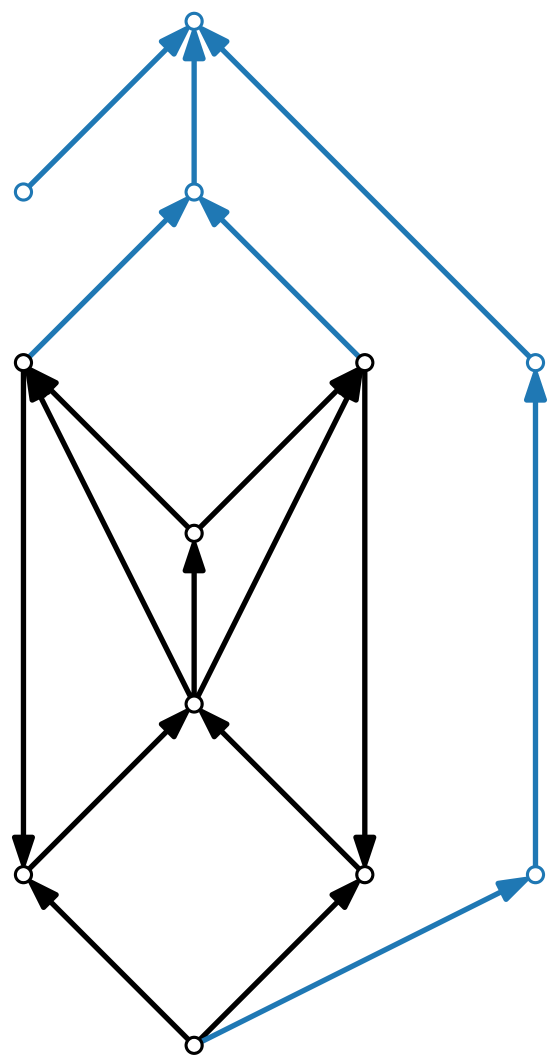
while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$

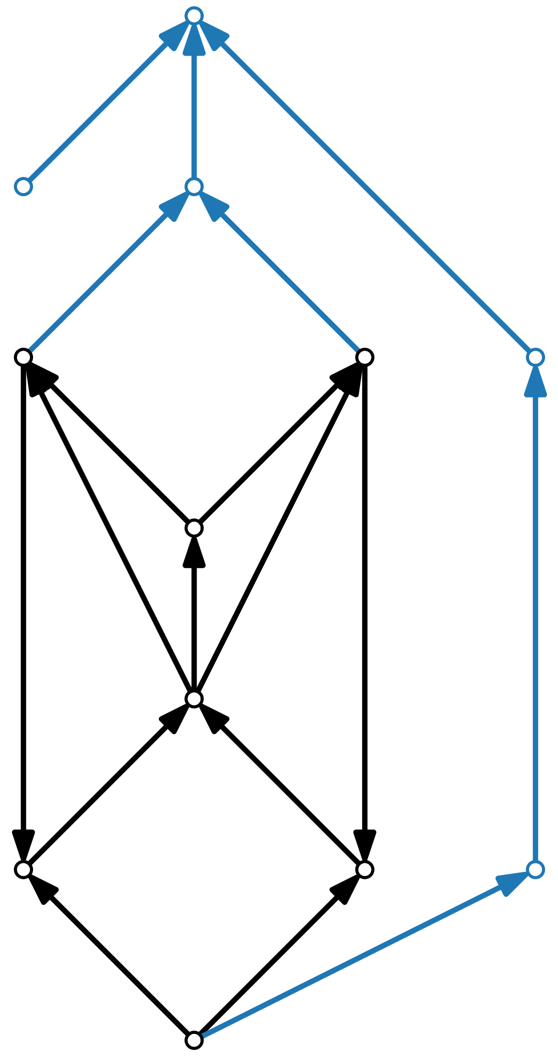
  Remove all isolated vertices from V



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$
  Remove all isolated vertices from V
  while in V exists a source v **do**



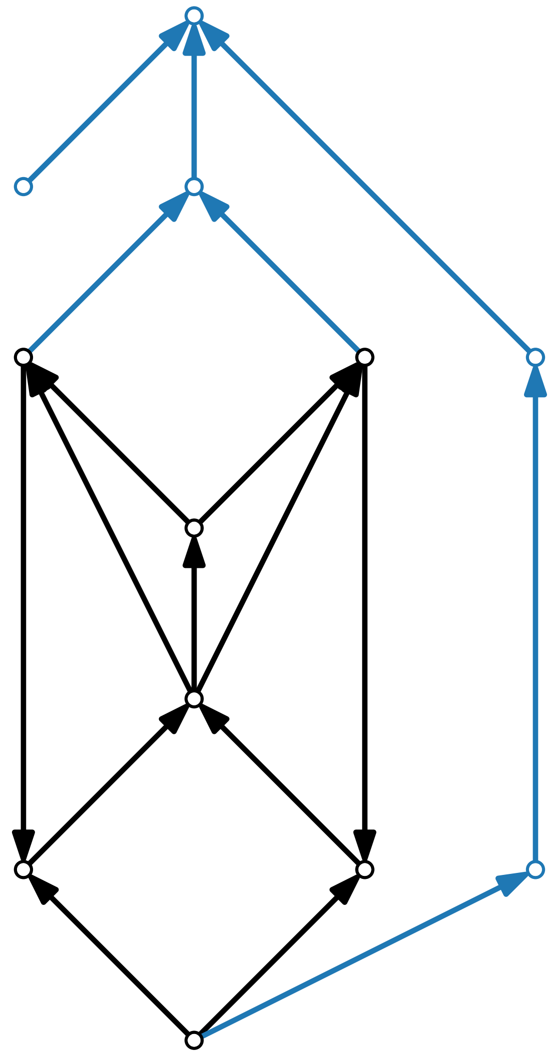
Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

- **while** in V exists a sink v **do**
  - $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  - remove v and $N^{\leftarrow}(v)$
- Remove all isolated vertices from V
- **while** in V exists a source v **do**
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N^{\rightarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

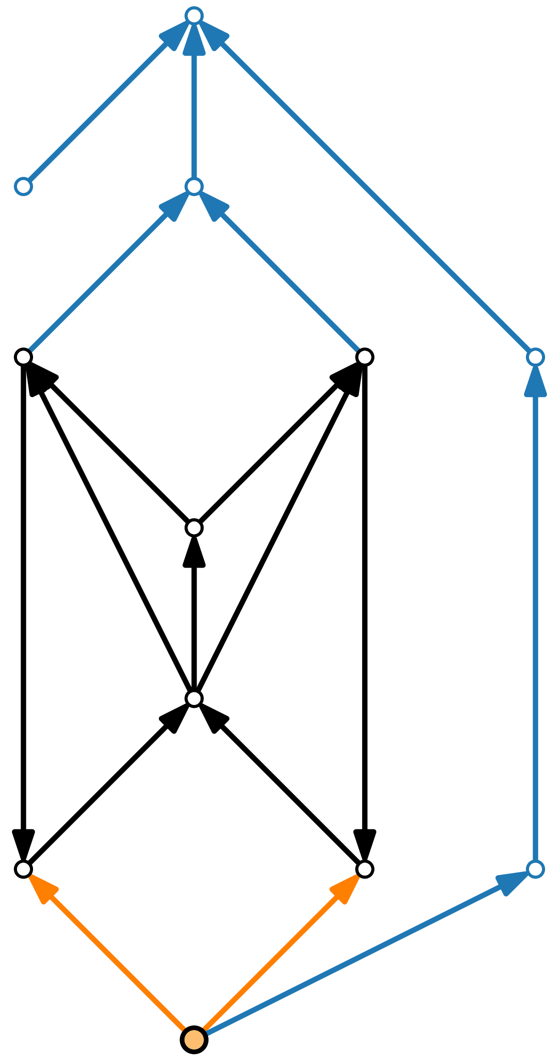
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

    remove v and $N^{\rightarrow}(v)$



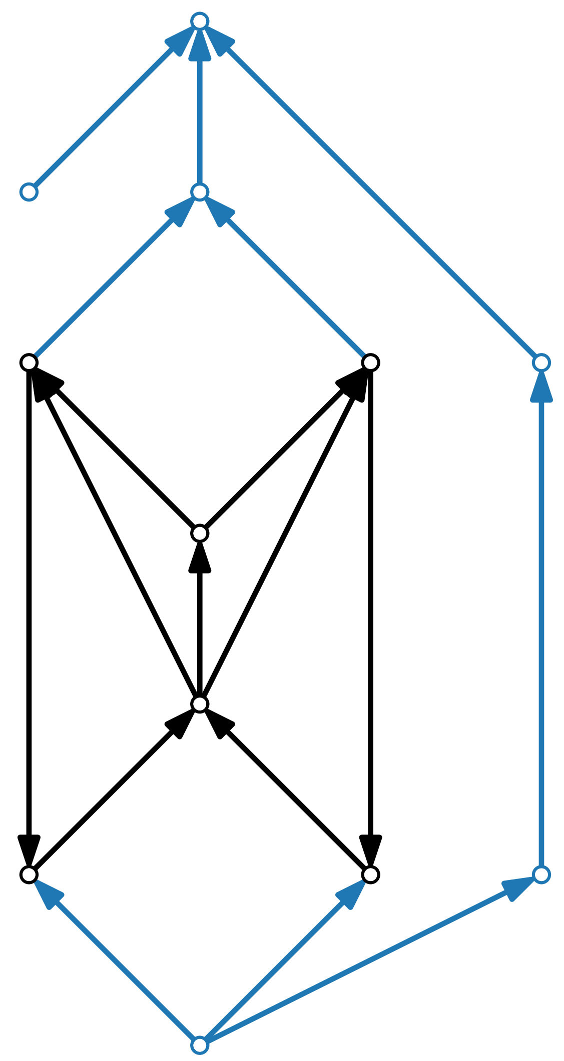
Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$

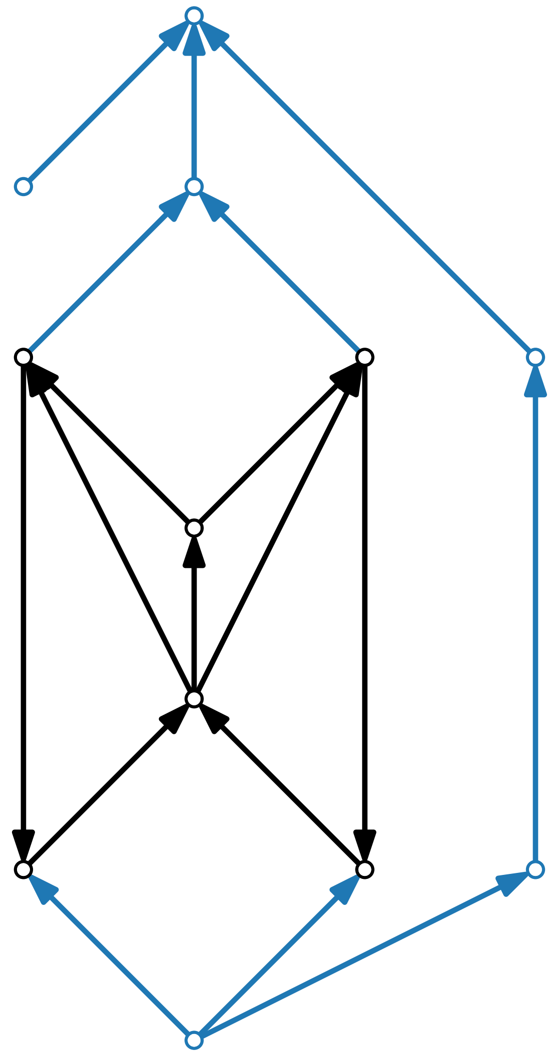
  Remove all isolated vertices from V

  while in V exists a source v **do**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

    remove v and $N^{\rightarrow}(v)$

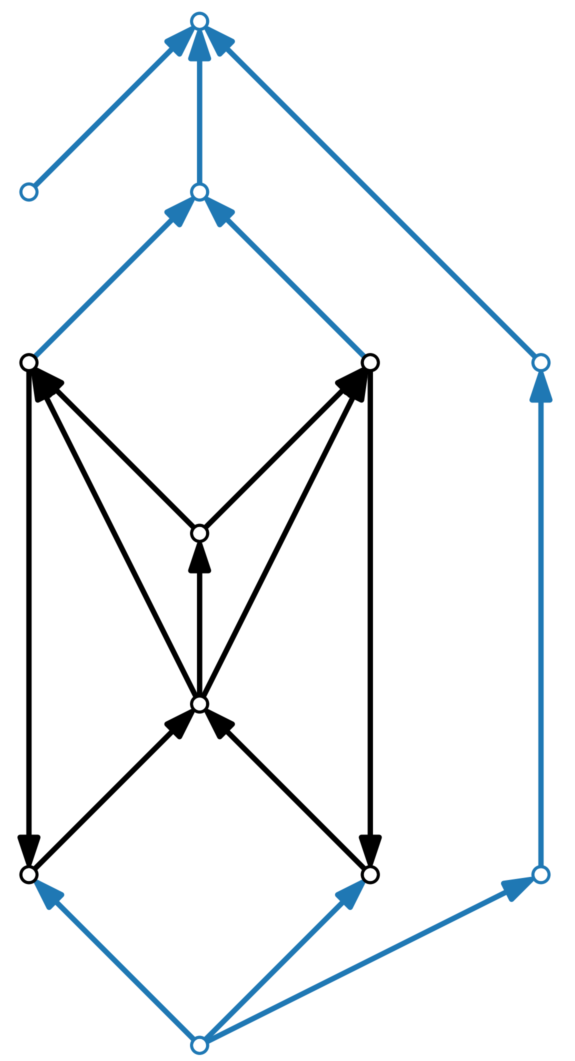
  if $V \neq \emptyset$ **then**



Heuristic 2

[Eades, Lin, Smyth '93]

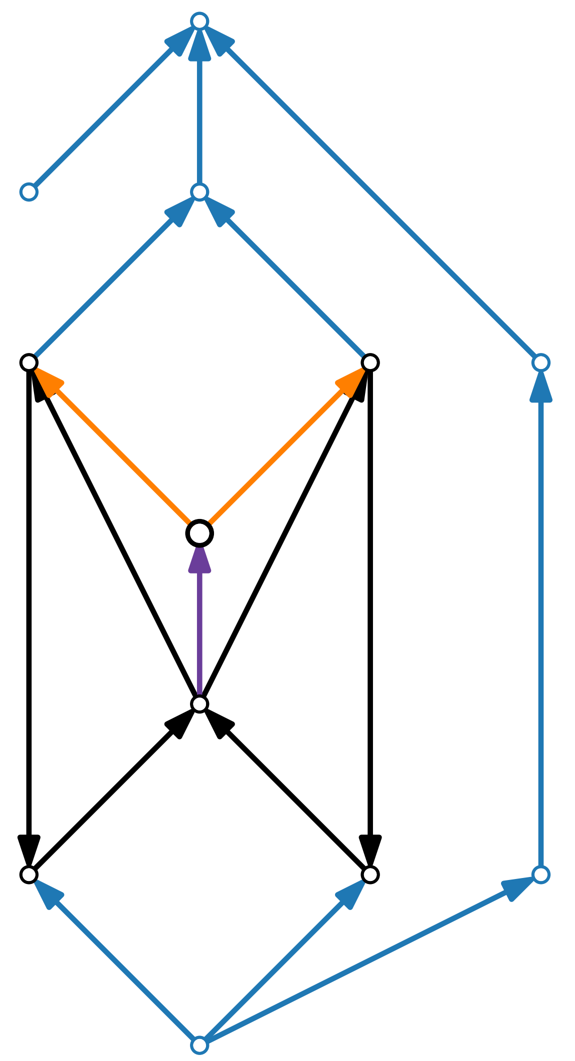
$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$
  Remove all isolated vertices from V
  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$
  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
- **while** in V exists a sink v **do**
  - $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  - remove v and $N^{\leftarrow}(v)$
- Remove all isolated vertices from V
- **while** in V exists a source v **do**
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N^{\rightarrow}(v)$
- **if** $V \neq \emptyset$ **then**
  - let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal



Heuristic 2

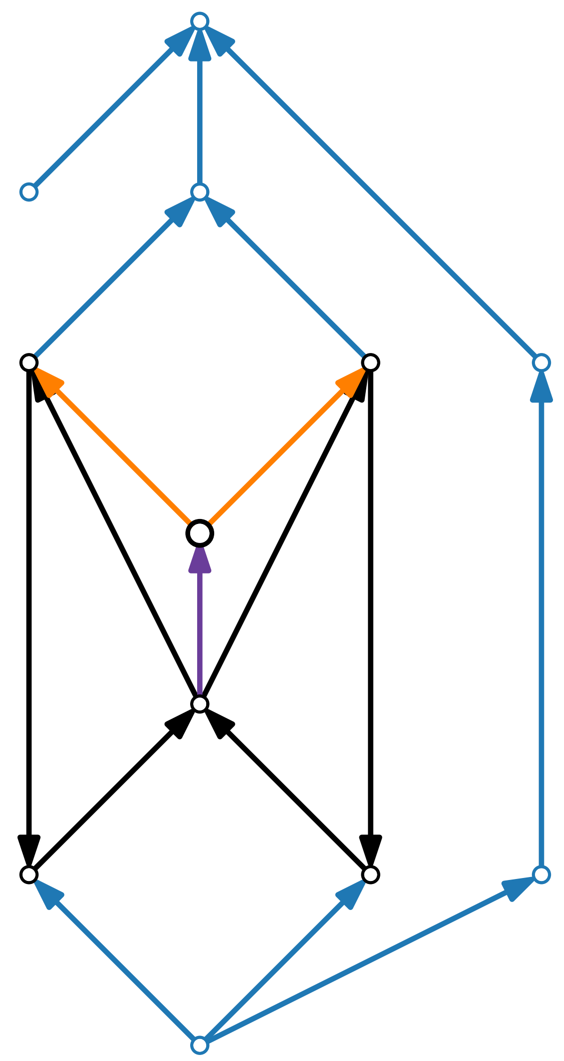
[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$

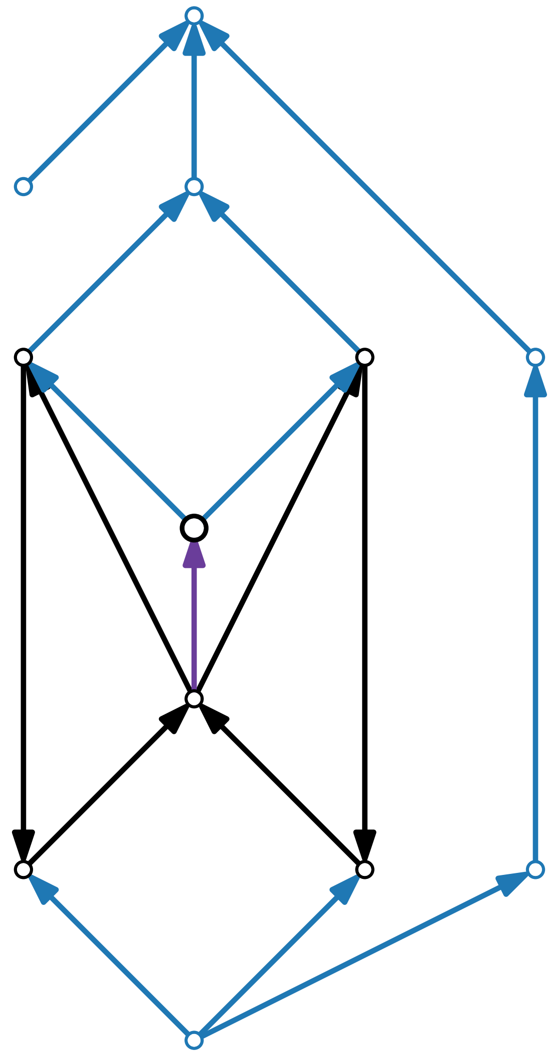
  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$
  Remove all isolated vertices from V
  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$
  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

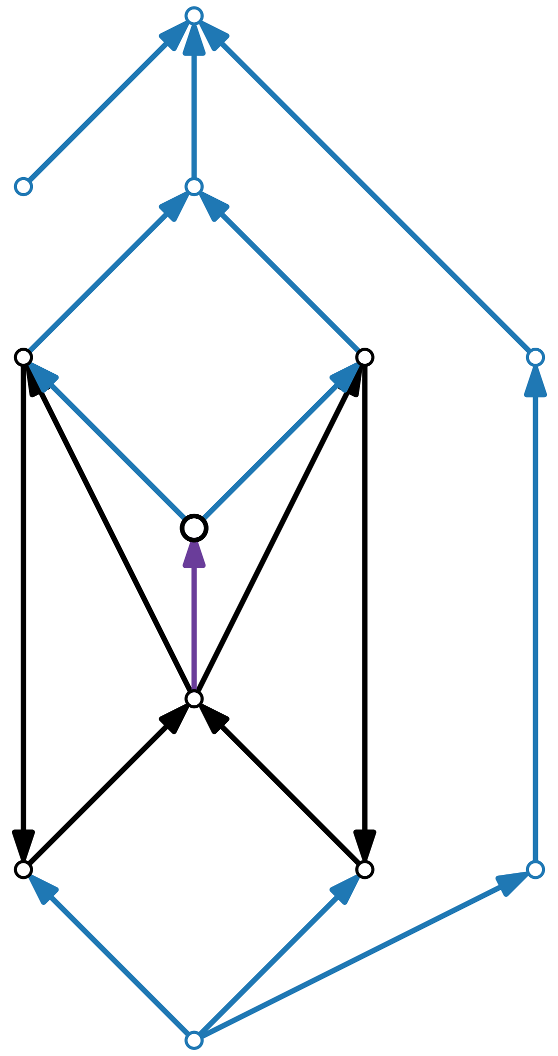
    remove v and $N^{\rightarrow}(v)$

  if $V \neq \emptyset$ **then**

    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

    remove v and $N(v)$



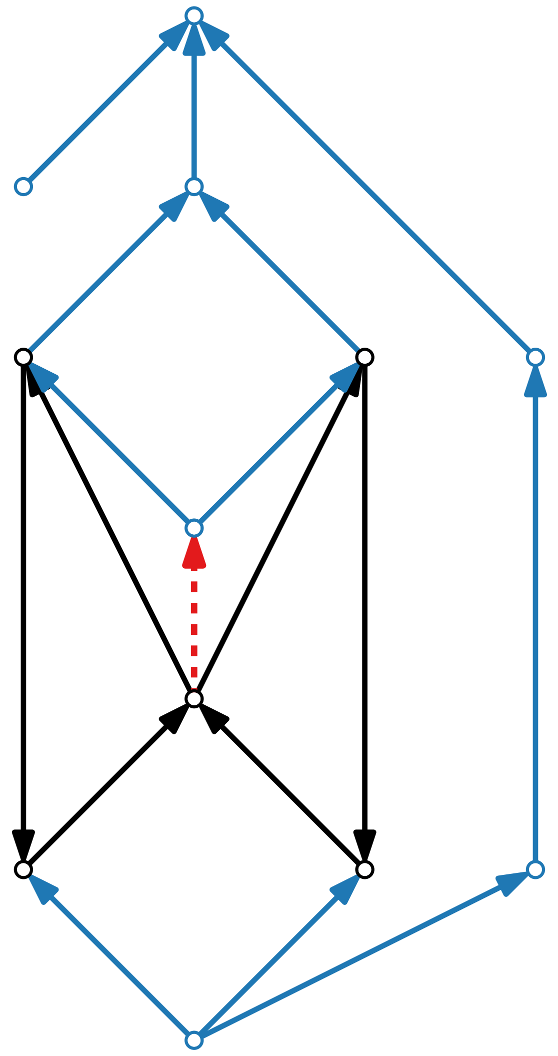
Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

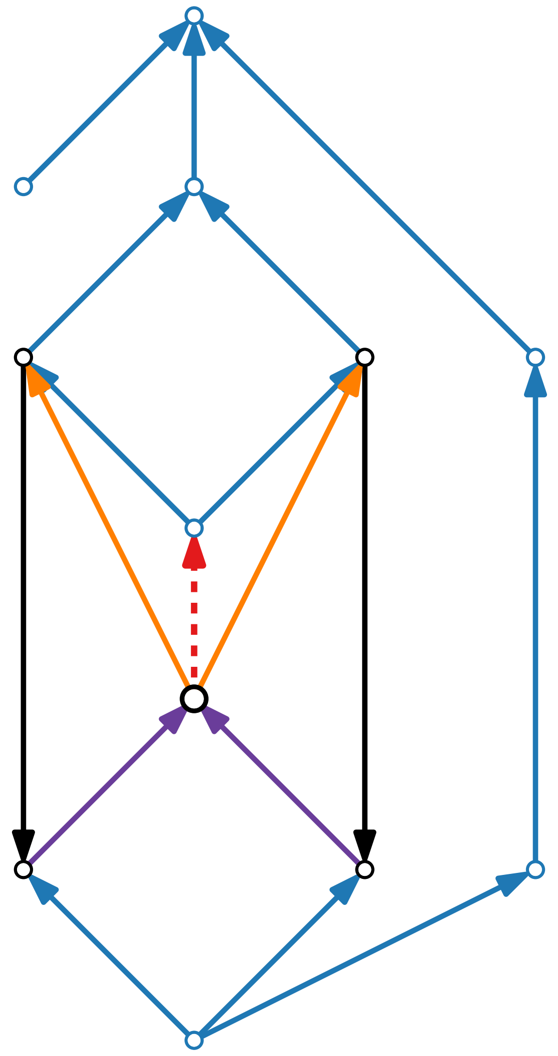
- **while** in V exists a sink v **do**
  - $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  - remove v and $N^{\leftarrow}(v)$
- Remove all isolated vertices from V
- **while** in V exists a source v **do**
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N^{\rightarrow}(v)$
- **if** $V \neq \emptyset$ **then**
  - let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

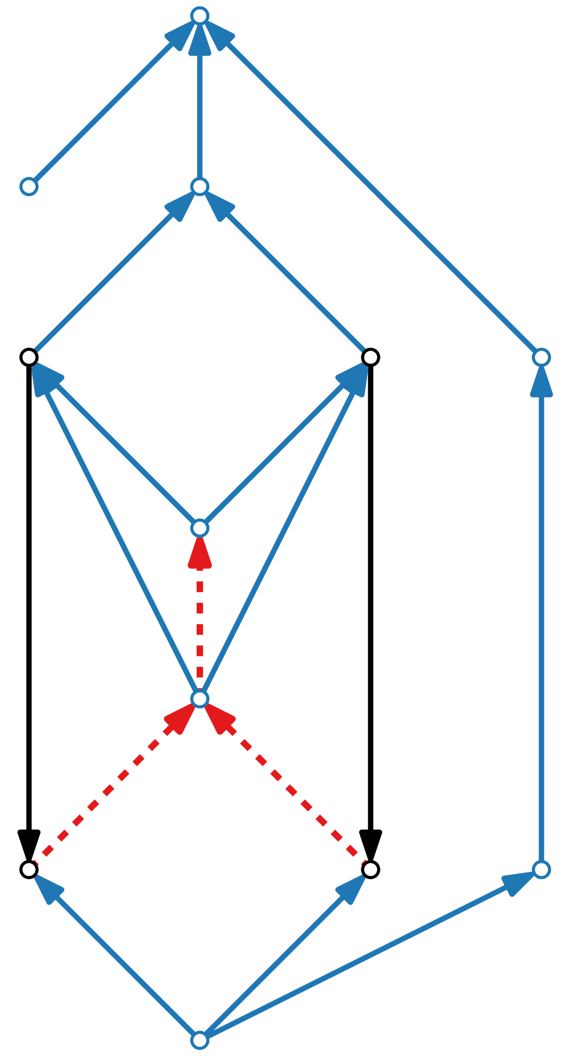
$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$
  Remove all isolated vertices from V
  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$
  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
    while in V exists a sink v **do**
        $E' \leftarrow E' \cup N^{\leftarrow}(v)$
        remove v and $N^{\leftarrow}(v)$
    Remove all isolated vertices from V
    while in V exists a source v **do**
        $E' \leftarrow E' \cup N^{\rightarrow}(v)$
        remove v and $N^{\rightarrow}(v)$
    if $V \neq \emptyset$ **then**
        let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
        $E' \leftarrow E' \cup N^{\rightarrow}(v)$
        remove v and $N(v)$



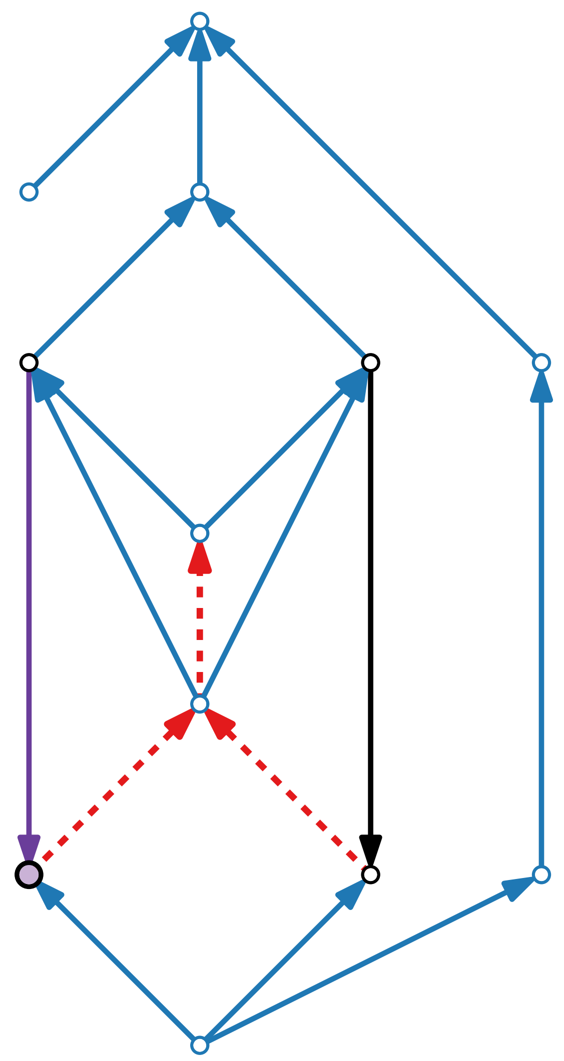
Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

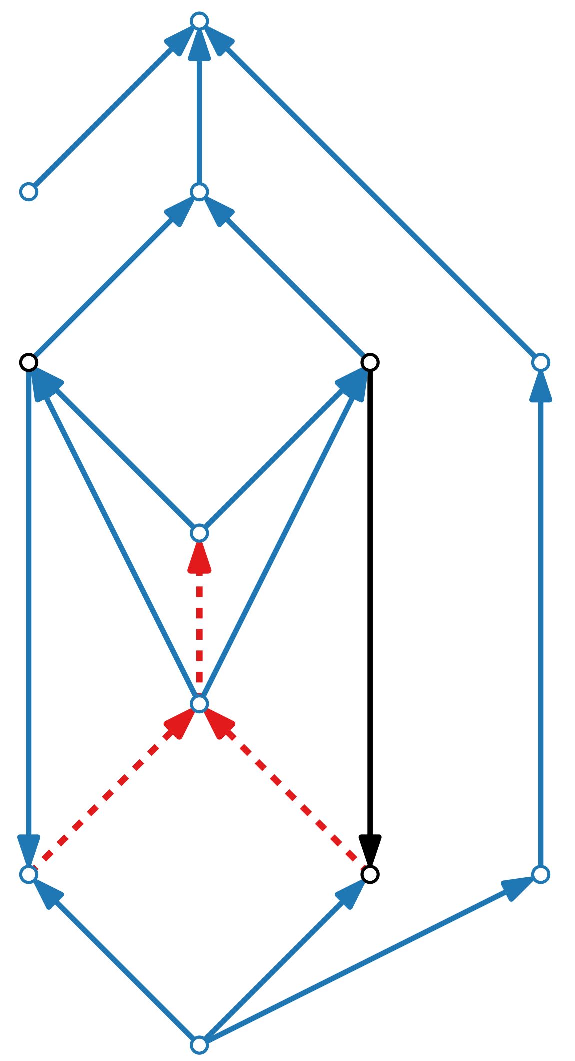
- **while** in V exists a sink v **do**
  - $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  - remove v and $N^{\leftarrow}(v)$
- Remove all isolated vertices from V
- **while** in V exists a source v **do**
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N^{\rightarrow}(v)$
- **if** $V \neq \emptyset$ **then**
  - let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

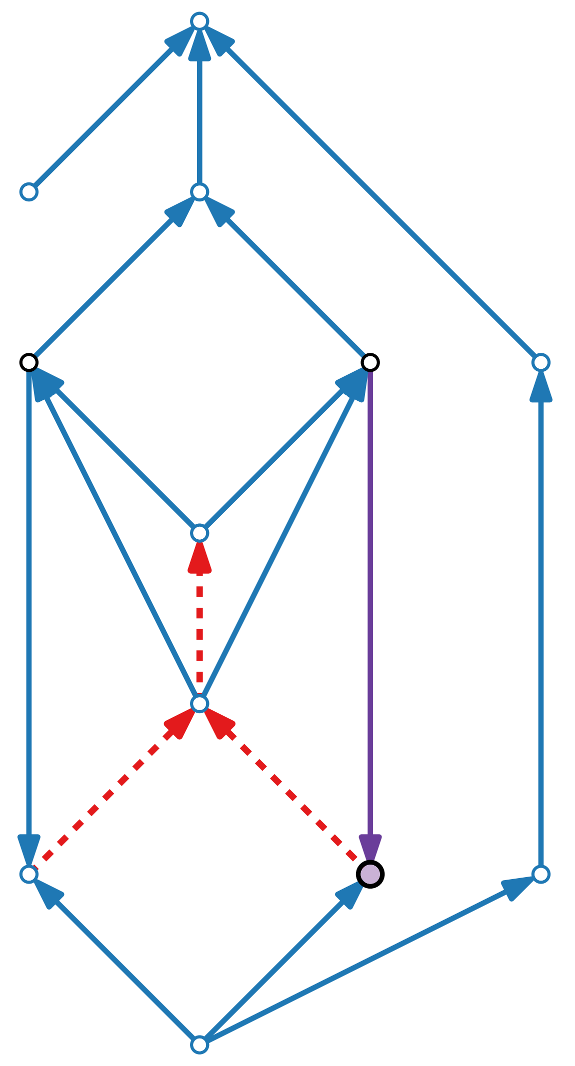
$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
- **while** in V exists a sink v **do**
  - $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  - remove v and $N^{\leftarrow}(v)$
- Remove all isolated vertices from V
- **while** in V exists a source v **do**
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N^{\rightarrow}(v)$
- **if** $V \neq \emptyset$ **then**
  - let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
- **while** in V exists a sink v **do**
  - $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  - remove v and $N^{\leftarrow}(v)$
- Remove all isolated vertices from V
- **while** in V exists a source v **do**
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N^{\rightarrow}(v)$
- **if** $V \neq \emptyset$ **then**
  - let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N(v)$



Heuristic 2

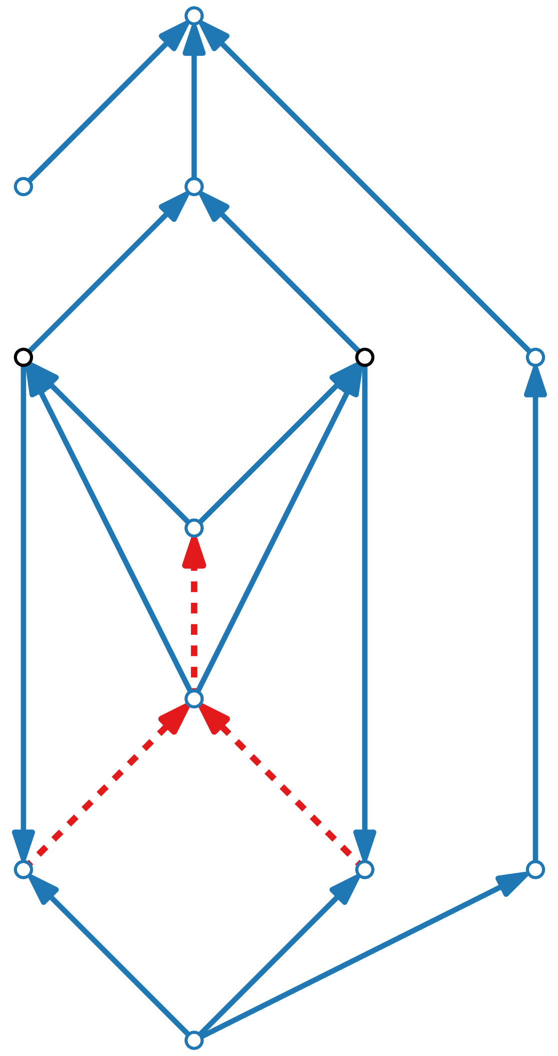
[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$

  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

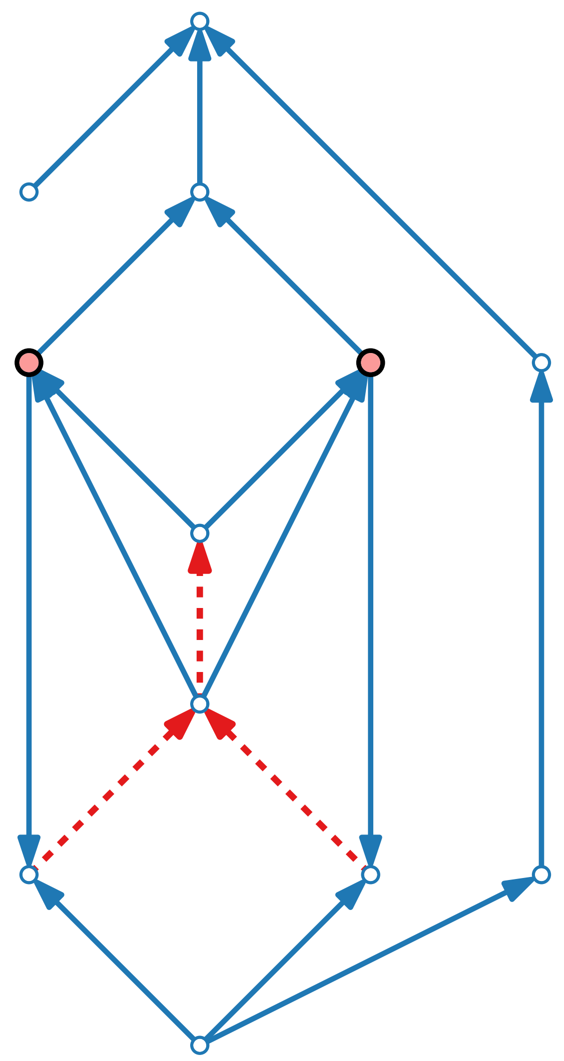
    remove v and $N^{\rightarrow}(v)$

  if $V \neq \emptyset$ **then**

    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

    remove v and $N(v)$



Heuristic 2

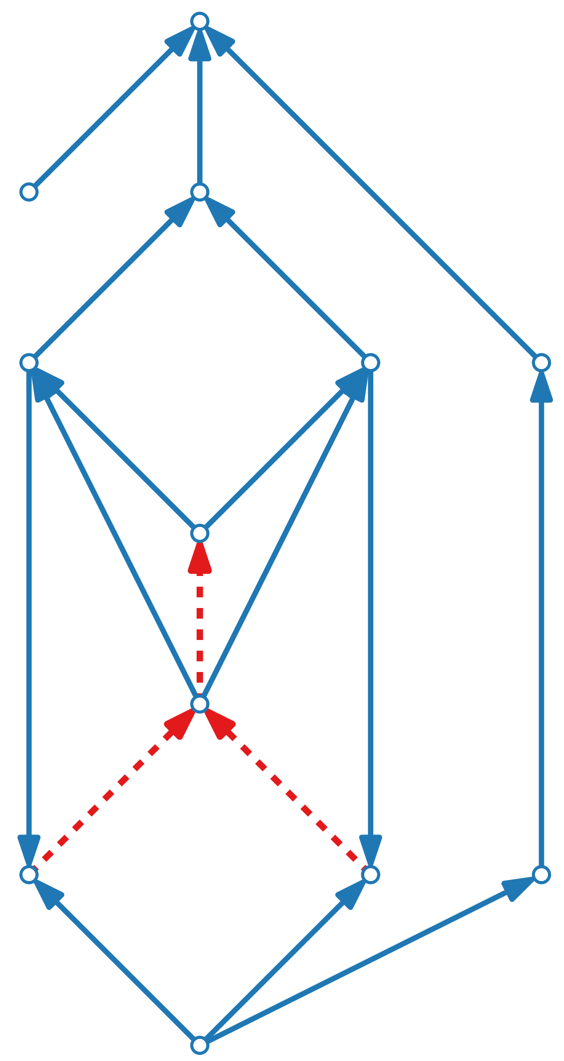
[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$

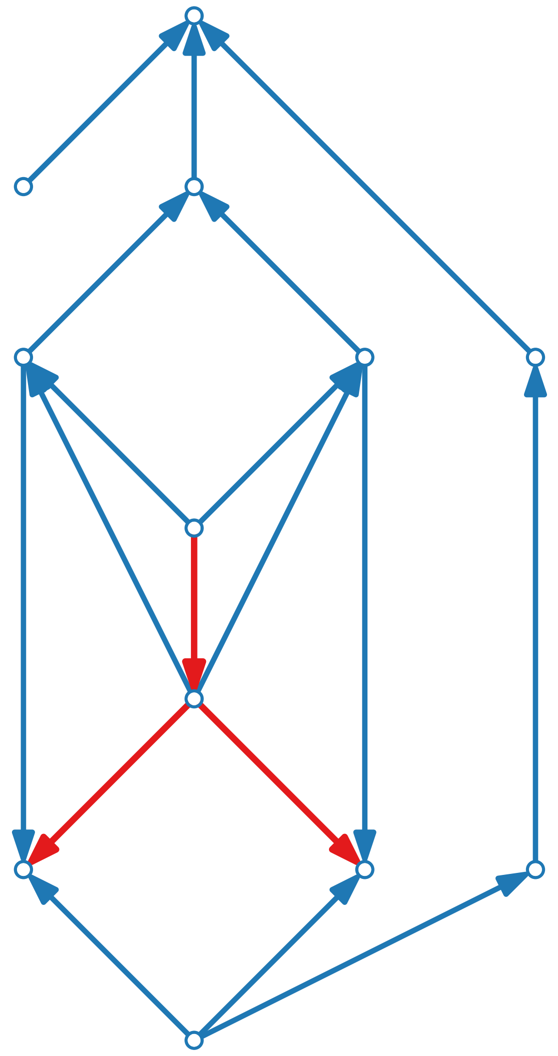
  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
- **while** in V exists a sink v **do**
  - $E' \leftarrow E' \cup N^{\leftarrow}(v)$
  - remove v and $N^{\leftarrow}(v)$
- Remove all isolated vertices from V
- **while** in V exists a source v **do**
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N^{\rightarrow}(v)$
- **if** $V \neq \emptyset$ **then**
  - let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
  - $E' \leftarrow E' \cup N^{\rightarrow}(v)$
  - remove v and $N(v)$



Heuristic 2

[Eades, Lin, Smyth '93]

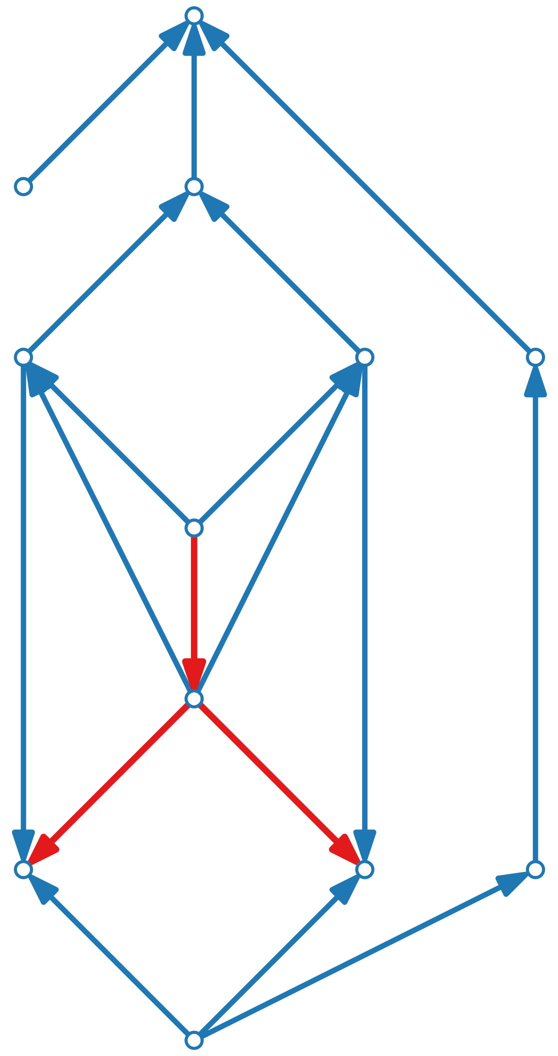
$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$

  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N(v)$

- Time: $\mathcal{O}(|V| + |E|)$



Heuristic 2

[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$

while $V \neq \emptyset$ **do**

  while in V exists a sink v **do**

    $E' \leftarrow E' \cup N^{\leftarrow}(v)$

    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

  while in V exists a source v **do**

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

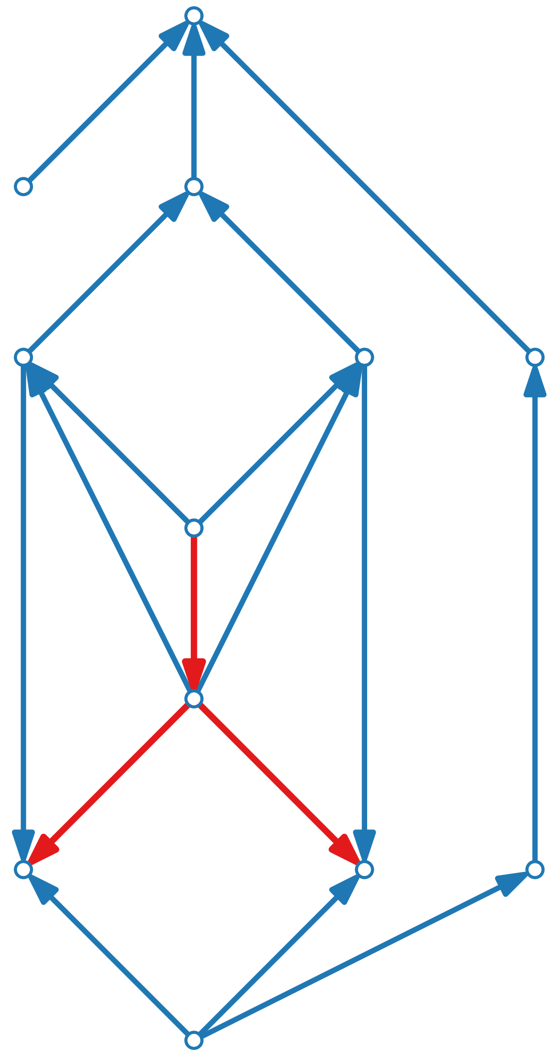
    remove v and $N^{\rightarrow}(v)$

  if $V \neq \emptyset$ **then**

    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal

    $E' \leftarrow E' \cup N^{\rightarrow}(v)$

    remove v and $N(v)$



- Time: $\mathcal{O}(|V| + |E|)$ [The main idea is to use bins for the sinks, sources, and a bin for each $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$]

Heuristic 2

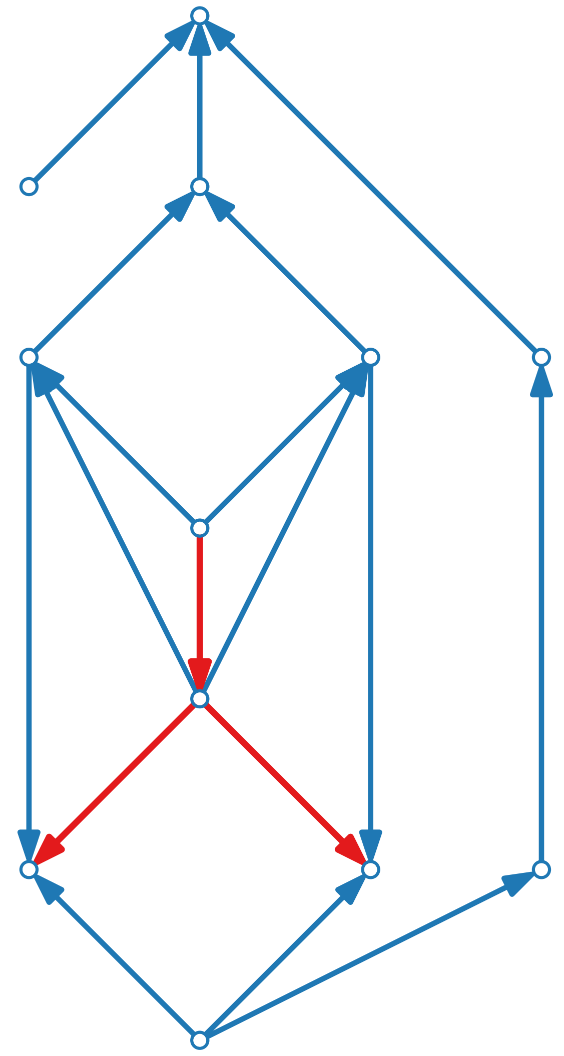
[Eades, Lin, Smyth '93]

$E' \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  while in V exists a sink v **do**
    $E' \leftarrow E' \cup N^{\leftarrow}(v)$
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated vertices from V

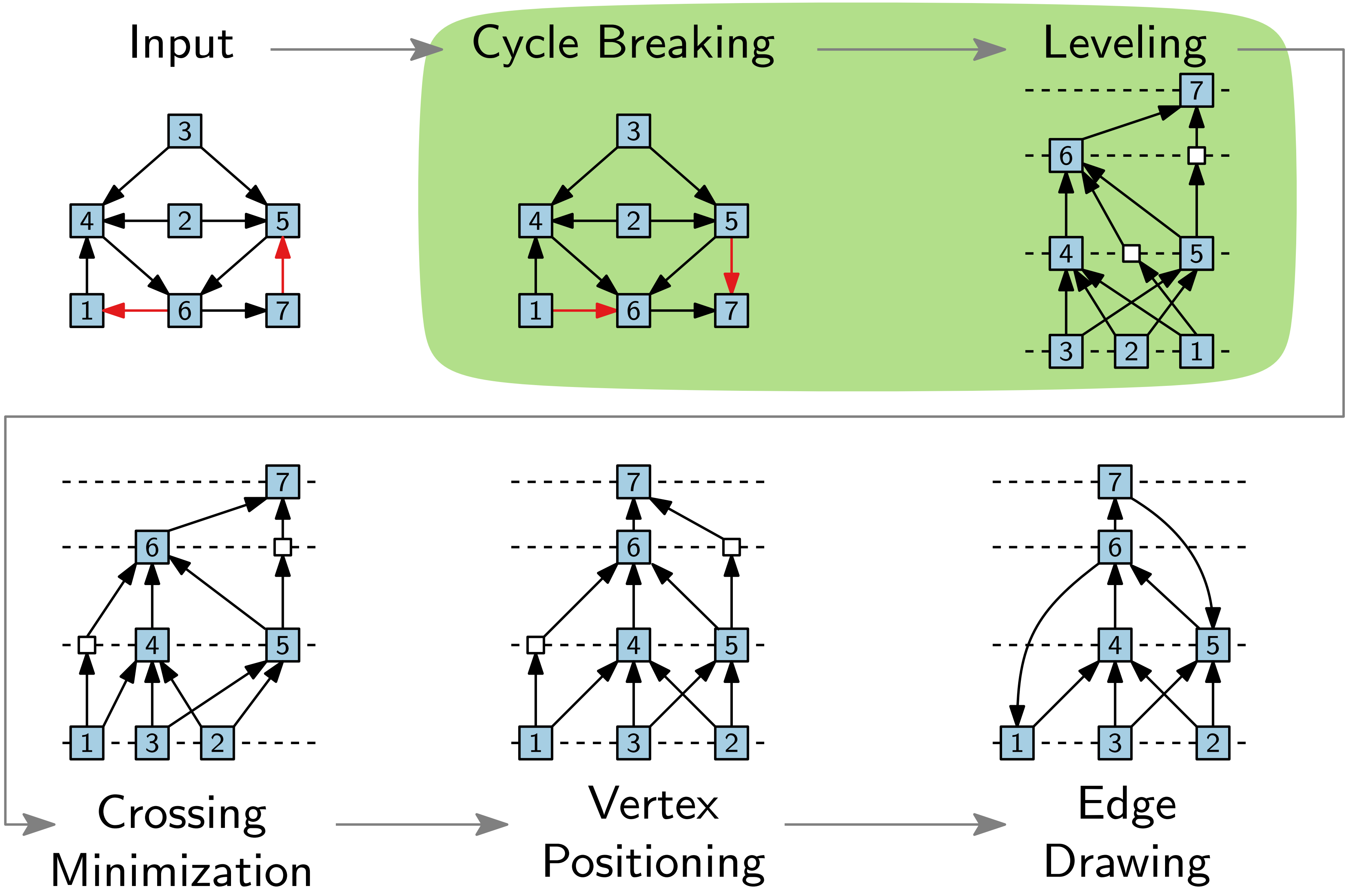
  while in V exists a source v **do**
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N^{\rightarrow}(v)$

  if $V \neq \emptyset$ **then**
    let $v \in V$ such that $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$ maximal
    $E' \leftarrow E' \cup N^{\rightarrow}(v)$
    remove v and $N(v)$

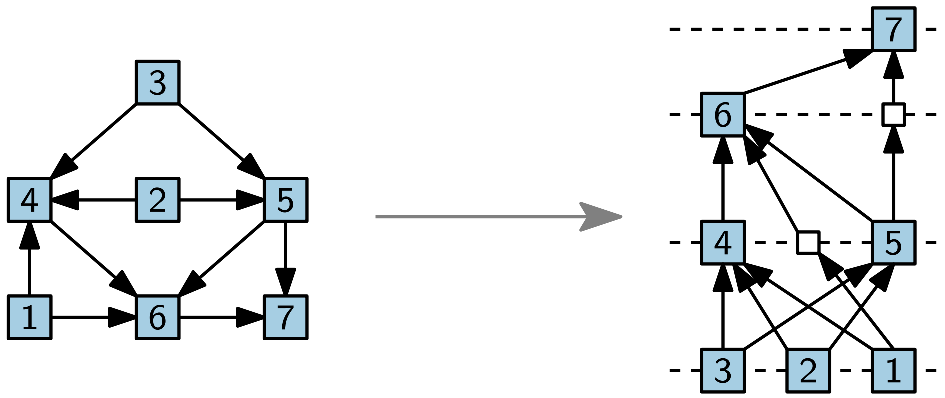


- Time: $\mathcal{O}(|V| + |E|)$ [The main idea is to use bins for the sinks, sources, and a bin for each $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$]
- Quality guarantee: $|E'| \geq |E|/2 + |V|/6$

Step 2: Leveling

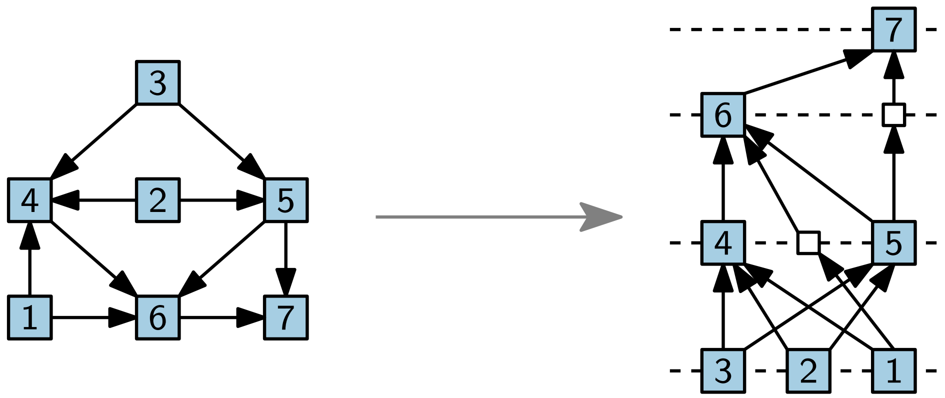


Step 2: Leveling



Problem.

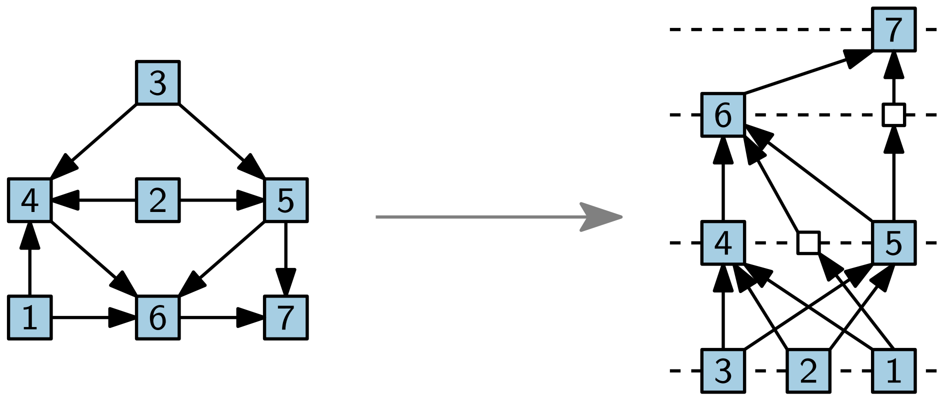
Step 2: Leveling



Problem.

- Input: acyclic digraph $G = (V, E)$

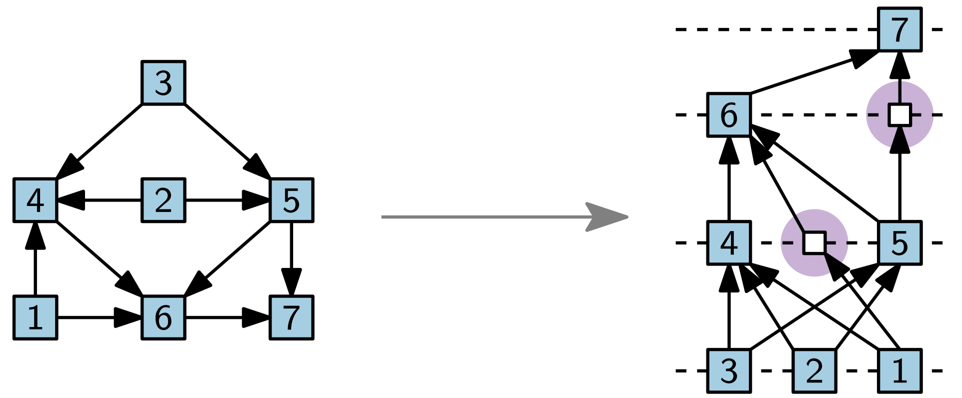
Step 2: Leveling



Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y\colon V \to \{1, \ldots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Step 2: Leveling

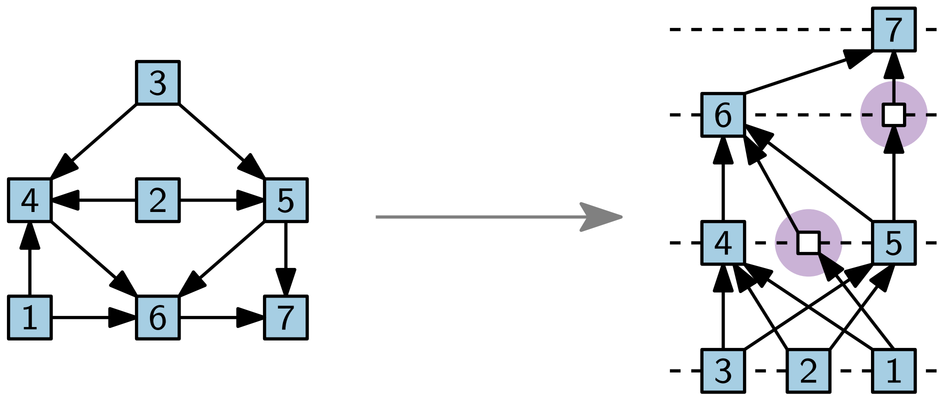


whenever an edge spans across a layer, we insert a *dummy vertex*

Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y \colon V \to \{1, \dots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Step 2: Leveling



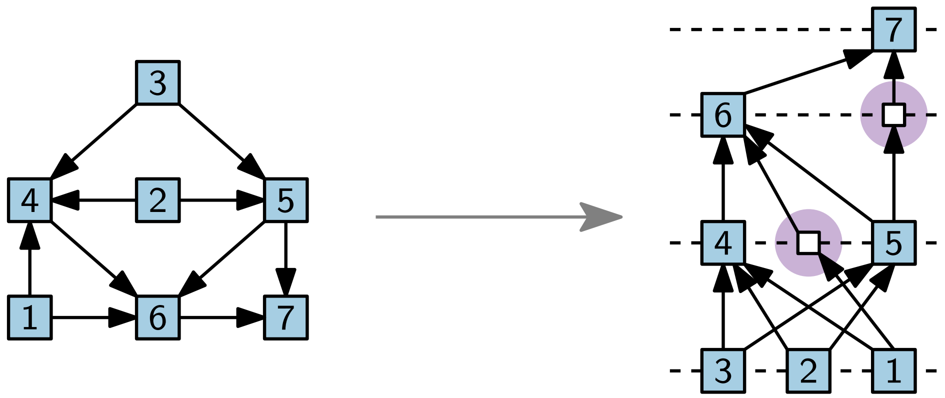
whenever an edge spans across a layer, we insert a *dummy vertex*

Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y\colon V \to \{1, \ldots, n\}$,
  such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize . . .*

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

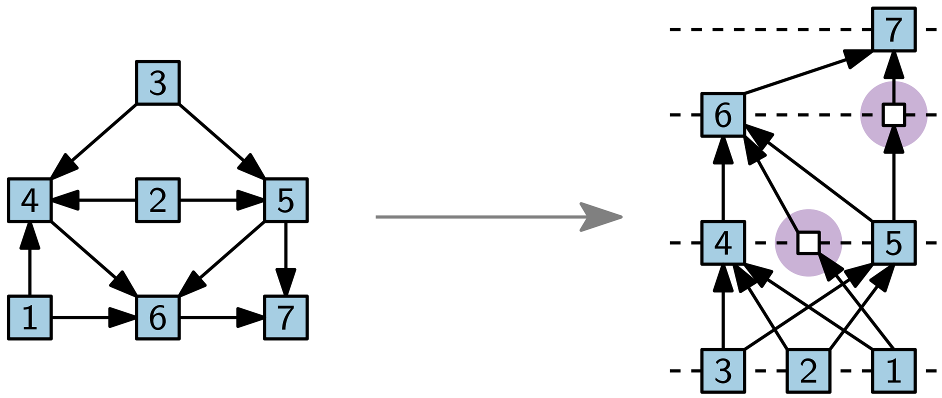
Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y\colon V \to \{1, \ldots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize . . .*

- number of layers,

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

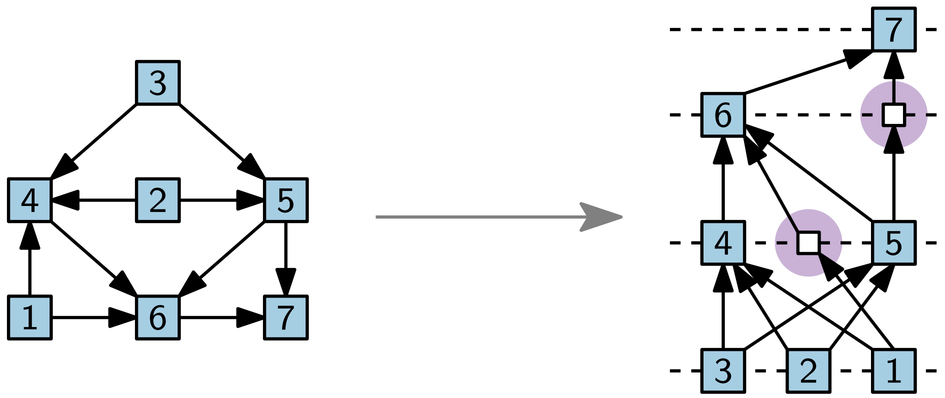
Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y \colon V \to \{1, \ldots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize* ...

- number of layers, i.e., $\max_{v \in V} y(v)$

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

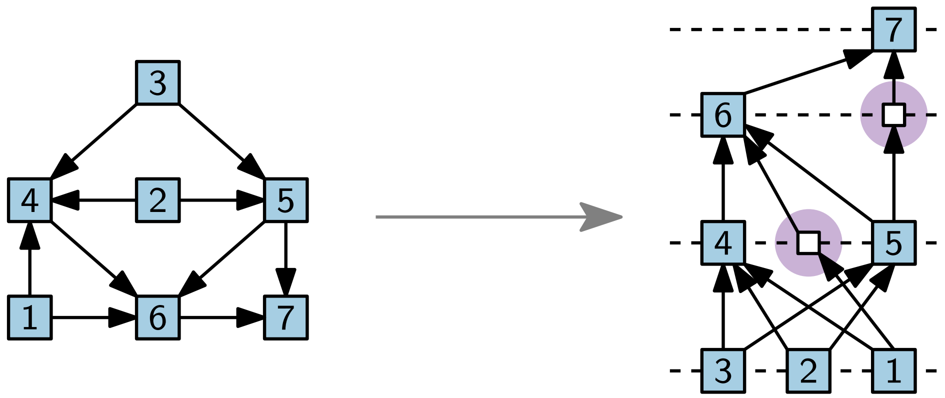
Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y \colon V \to \{1, \ldots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize . . .*

- number of layers, i.e., $\max_{v \in V} y(v)$
- length of the longest edge,

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

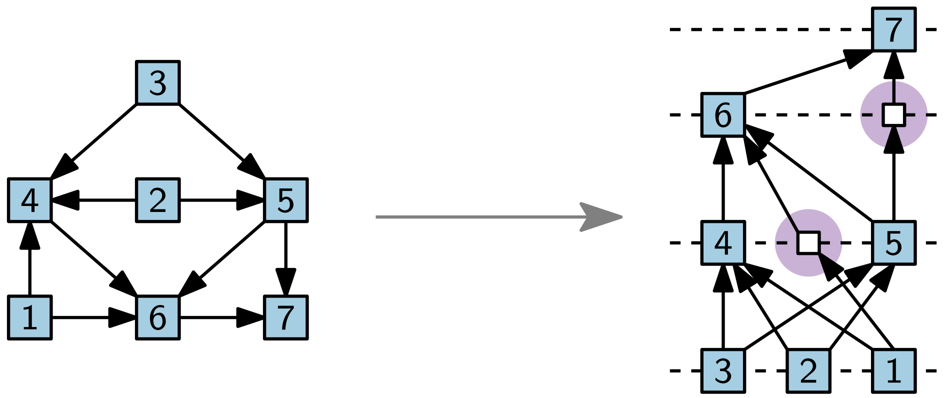
Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y\colon V \to \{1, \ldots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize . . .*

- number of layers, i.e., $\max_{v \in V} y(v)$
- length of the longest edge, i.e. $\max_{uv \in E} y(v) - y(u)$

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

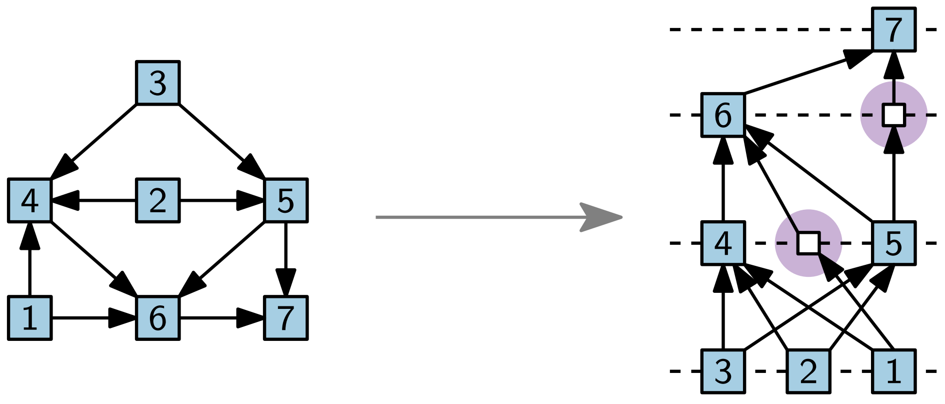
Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y\colon V \to \{1, \dots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize ...*

- number of layers, i.e., $\max_{v \in V} y(v)$
- length of the longest edge, i.e. $\max_{uv \in E} y(v) - y(u)$
- width,

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

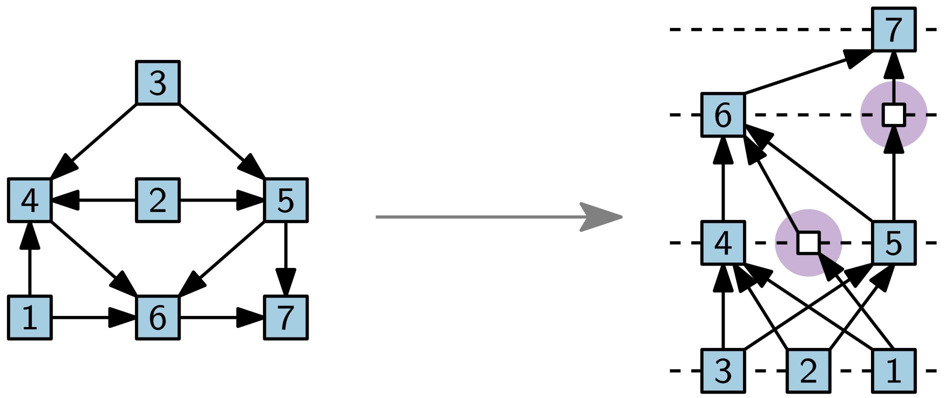
Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y \colon V \to \{1, \ldots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize . . .*

- number of layers, i.e., $\max_{v \in V} y(v)$
- length of the longest edge, i.e. $\max_{uv \in E} y(v) - y(u)$
- width, i.e., $\max_{i \in \{1,\ldots,n\}} |\{v \mid y(v) = i\}|$

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

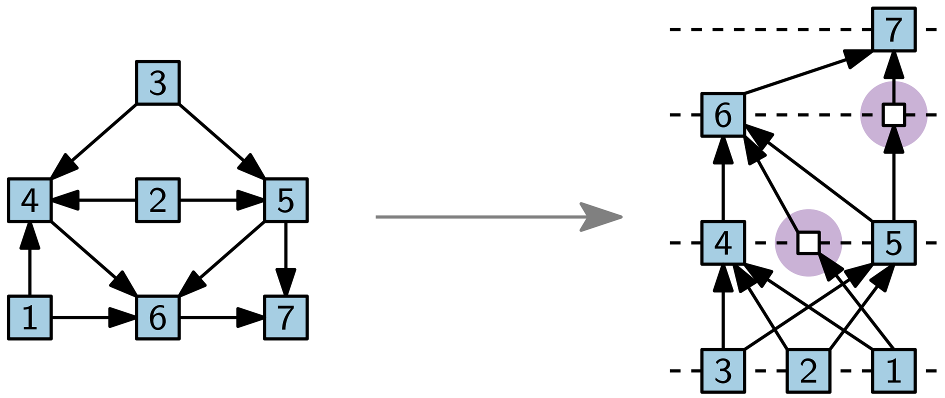
Problem.

- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y \colon V \to \{1, \ldots, n\}$, such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize . . .*

- number of layers, i.e., $\max_{v \in V} y(v)$
- length of the longest edge, i.e. $\max_{uv \in E} y(v) - y(u)$
- width, i.e., $\max_{i \in \{1,\ldots,n\}} |\{v \mid y(v) = i\}|$
- total edge length,

Step 2: Leveling



whenever an edge spans across a layer, we insert a *dummy vertex*

Problem.

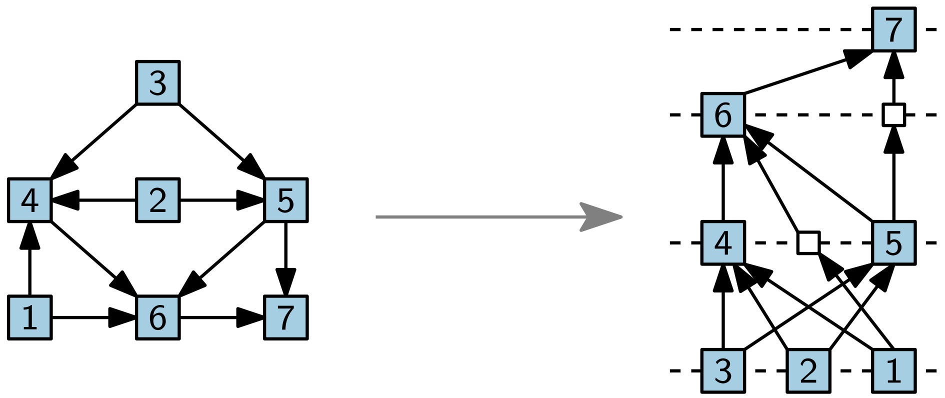
- Input: acyclic digraph $G = (V, E)$
- Output: Mapping $y\colon V \to \{1, \ldots, n\}$,
  such that for every $uv \in E$, $y(u) < y(v)$.

Objective is to *minimize . . .*

- number of layers, i.e., $\max_{v \in V} y(v)$
- length of the longest edge, i.e. $\max_{uv \in E} y(v) - y(u)$
- width, i.e., $\max_{i \in \{1,\ldots,n\}} |\{v \mid y(v) = i\}|$
- total edge length, i.e., number of dummy vertices

Minimize Number of Layers

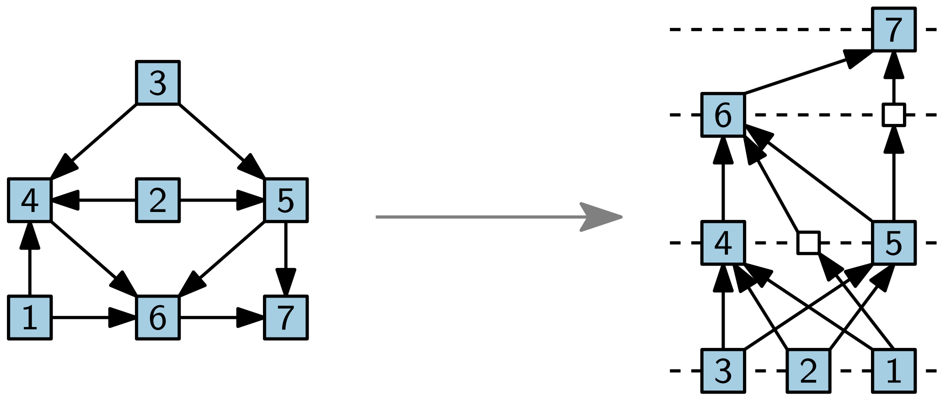
Algorithm.



Minimize Number of Layers

Algorithm.

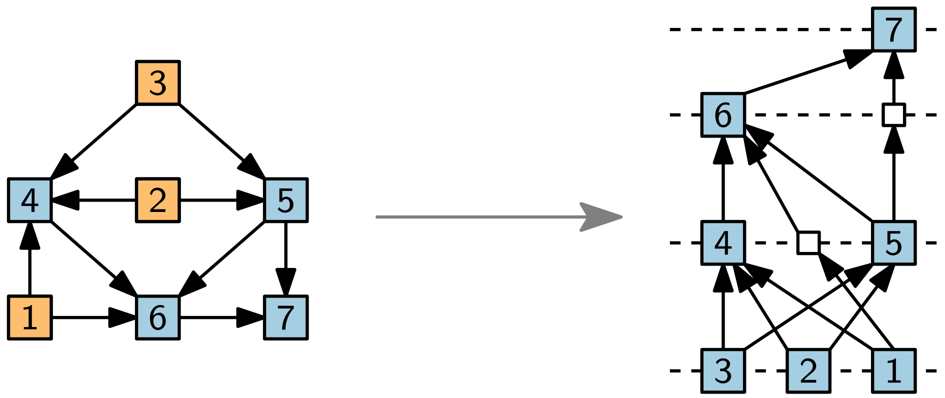
- for each source q
  set $y(q) := 1$



Minimize Number of Layers

Algorithm.

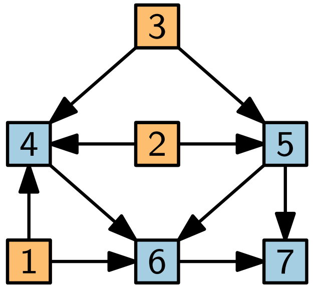
- for each source q
  set $y(q) := 1$

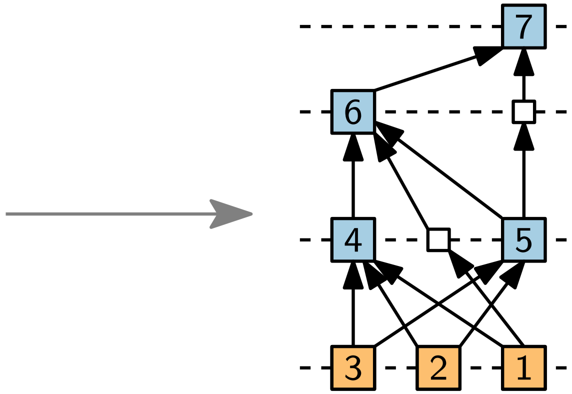


Minimize Number of Layers

Algorithm.

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

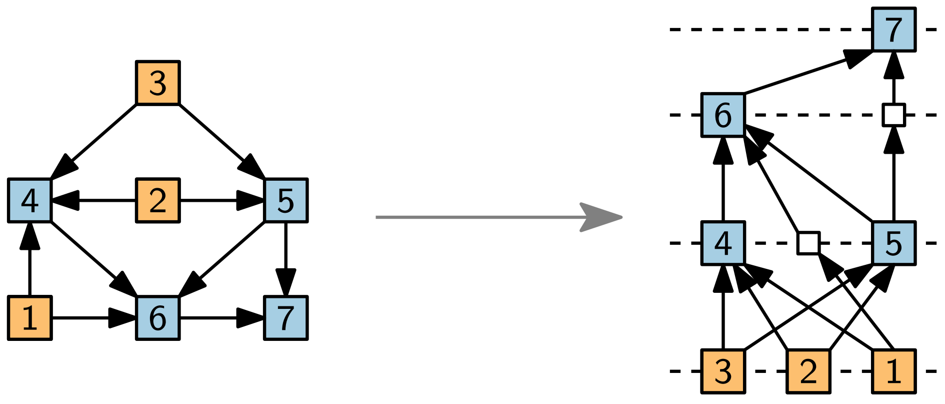




Minimize Number of Layers

Algorithm.

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

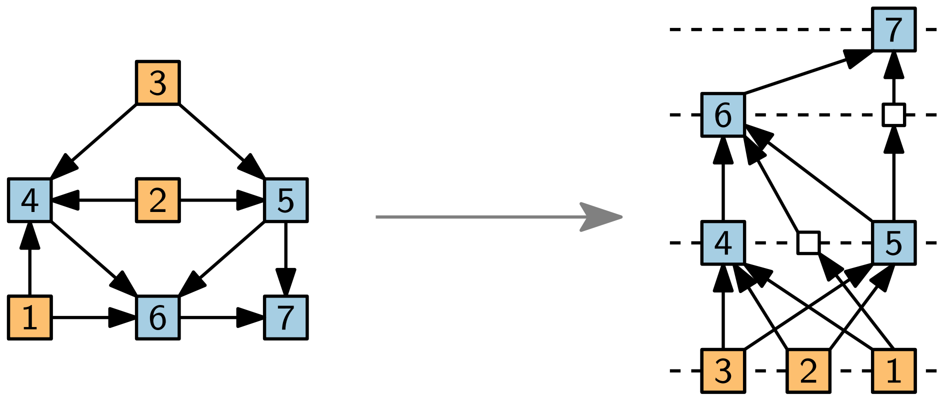


Minimize Number of Layers

Algorithm.

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

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



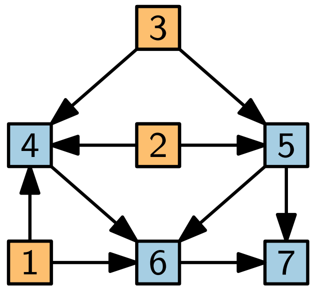
Observation.

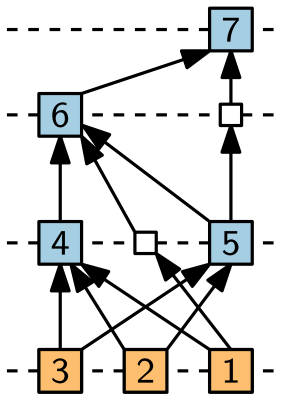
- $y(v)$ is

Minimize Number of Layers

Algorithm.

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





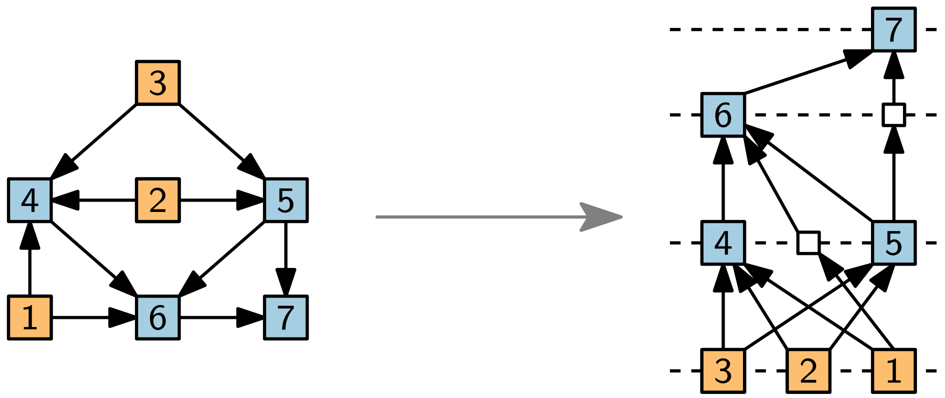
Observation.

- $y(v)$ is length of the longest path from a source to v plus 1.

Minimize Number of Layers

Algorithm.

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



Observation.

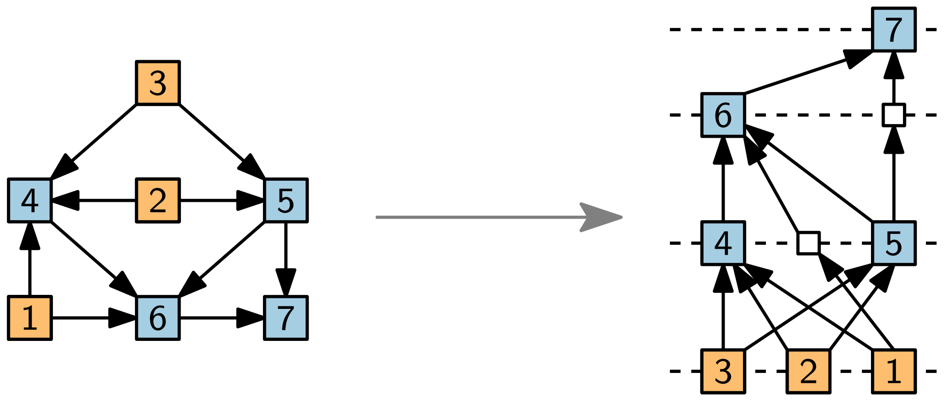
- $y(v)$ is length of the longest path from a source to v plus 1.

  ...which is optimal!

Minimize Number of Layers

Algorithm.

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

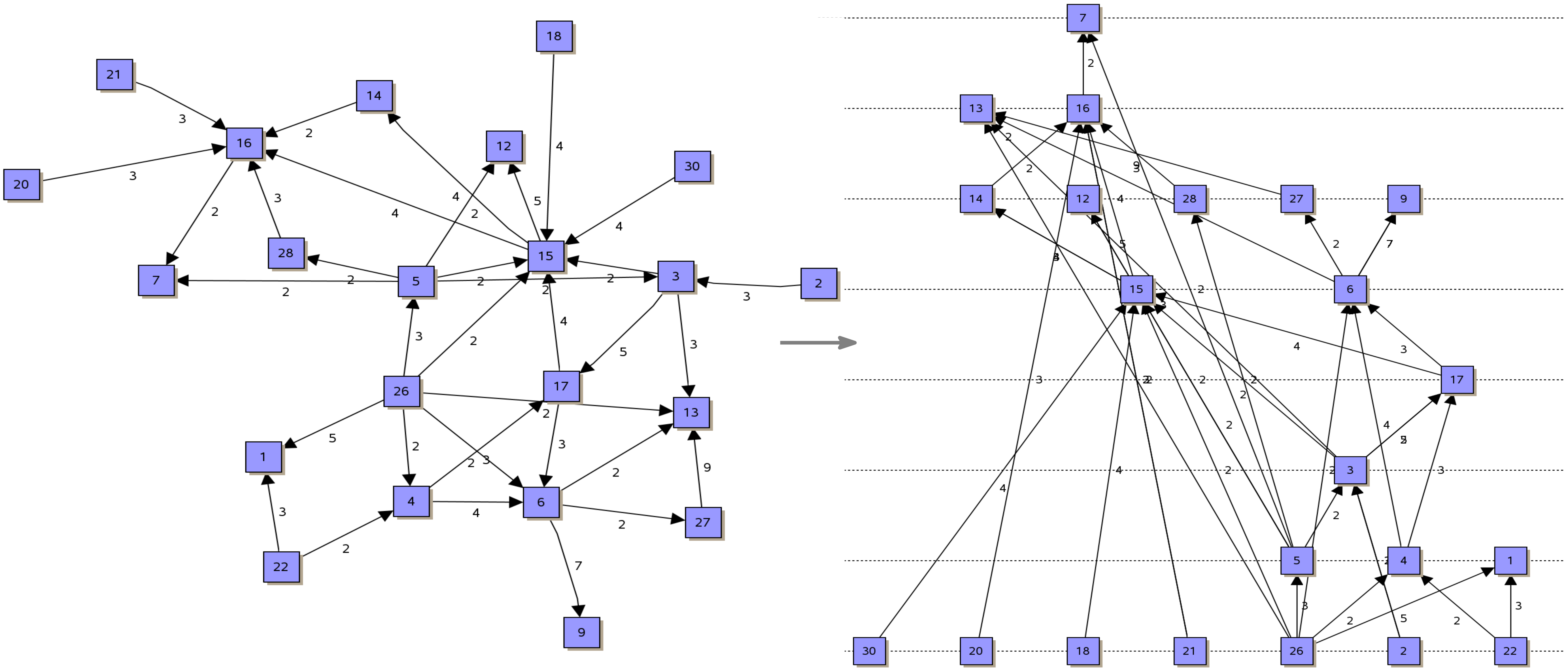


Observation.

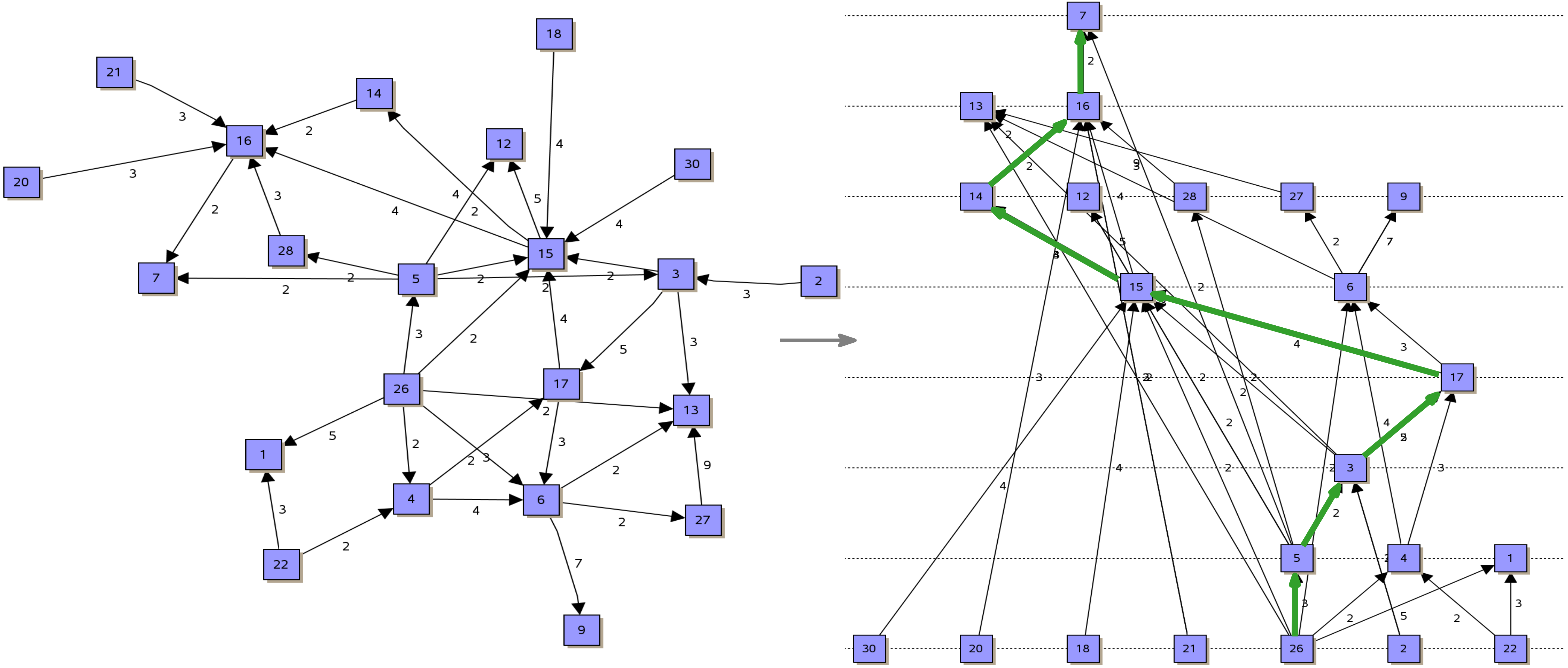
- $y(v)$ is length of the longest path from a source to v plus 1.

  ... which is optimal!
- Can be implemented in linear time with recursive algorithm.

Example



Example



21
14
18
16
12
30
20
28
15
7
5
3
2
26
17
13
1
4
6
27
22
9
7
13
16
14
12
28
27
9
15
6
17
3
5
4
1
30
20
18
21
26
2
22

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$\text{min} \quad \sum_{(u,v)\in E}(y(v) - y(u))$$

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$\begin{array}{rl} \text{min} & \sum_{(u,v)\in E}(y(v)-y(u)) \\ \text{subject to} & \end{array}$$

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$\begin{array}{rll} \text{min} & \sum_{(u,v)\in E}(y(v) - y(u)) & \\ \text{subject to} & y(v) - y(u) \geq 1 & \forall (u,v) \in E \end{array}$$

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$
\begin{array}{rll}
\text{min} & \sum_{(u,v)\in E}(y(v)-y(u)) & \\
\text{subject to} & y(v)-y(u)\geq 1 & \forall (u,v)\in E \\
 & y(v)\geq 1 & \forall v\in V
\end{array}
$$

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$
\begin{array}{rlr}
\text{min} & \sum_{(u,v)\in E}(y(v)-y(u)) & \\
\text{subject to} & y(v)-y(u)\geq 1 & \forall (u,v)\in E \\
& y(v)\geq 1 & \forall v\in V \\
& y(v)\in \mathbb{Z} & \forall v\in V
\end{array}
$$

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$\begin{array}{rll} \text{min} & \sum_{(u,v)\in E}(y(v)-y(u)) & \\ \text{subject to} & y(v)-y(u)\geq 1 & \forall (u,v)\in E \\ & y(v)\geq 1 & \forall v\in V \\ & y(v)\in \mathbb{Z} & \forall v\in V \end{array}$$

One can show that:

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$\begin{array}{rll}
\text{min} & \sum_{(u,v)\in E}(y(v)-y(u)) & \\
\text{subject to} & y(v)-y(u)\geq 1 & \forall (u,v)\in E \\
 & y(v)\geq 1 & \forall v\in V \\
 & y(v)\in\mathbb{Z} & \forall v\in V
\end{array}$$

One can show that:

- Constraint-matrix is **totally unimodular**.

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

$$\begin{array}{rll} \text{min} & \sum_{(u,v)\in E}(y(v) - y(u)) & \\ \text{subject to} & y(v) - y(u) \geq 1 & \forall (u,v) \in E \\ & y(v) \geq 1 & \forall v \in V \\ & y(v) \in \mathbb{Z} & \forall v \in V \end{array}$$

One can show that:

- Constraint-matrix is **totally unimodular**.

  $\Rightarrow$ Solution of the relaxed linear program is integer.

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

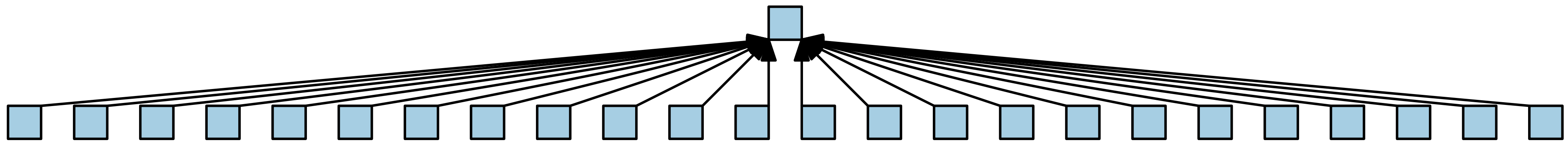
$$\begin{array}{rll} \text{min} & \sum_{(u,v)\in E}(y(v)-y(u)) & \\ \text{subject to} & y(v)-y(u)\geq 1 & \forall (u,v)\in E \\ & y(v)\geq 1 & \forall v\in V \\ & y(v)\in \mathbb{Z} & \forall v\in V \end{array}$$

One can show that:

- Constraint-matrix is **totally unimodular**.

  $\Rightarrow$ Solution of the relaxed linear program is integer.
- The total edge length can be minimized in polynomial time.

Width



Drawings can be very wide.

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

Problem: precedence-constrained multi-processor scheduling.

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

Problem: precedence-constrained multi-processor scheduling.

- Input: n jobs with unit (1) processing time, W identical machines, and a partial ordering $<$ on the jobs.

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

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 completion time (known as *makespan*).

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

same!

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 completion time (known as *makespan*).

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

same!

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

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 completion time (known as *makespan*).

- NP-hard,

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

same!

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 completion time (known as *makespan*).

- NP-hard, $(2 - \frac{1}{W})$-Approx.,

Narrower Layer Assignment

Problem: leveling with a given maximum-width.

- Input: acyclic digraph $G = (V, E)$, width $W > 0$
- Output: assignment of the vertices in V to layers, such that
  – the assignment is a leveling,
  – each layer contains at most W elements, and
  – the number of layers is minimized

same!

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 completion time (known as *makespan*).

- NP-hard, $(2 - \frac{1}{W})$-Approx., no $(\frac{4}{3} - \varepsilon)$-Approx. ($W \geq 3$)

Approximating PCMPS

- jobs stored in a list L
  (e.g., topologically sorted)

Approximating PCMPS

- jobs stored in a list L
  (e.g., topologically sorted)
- a job in L is *available* when all its predecessors have been scheduled

Approximating PCMPS

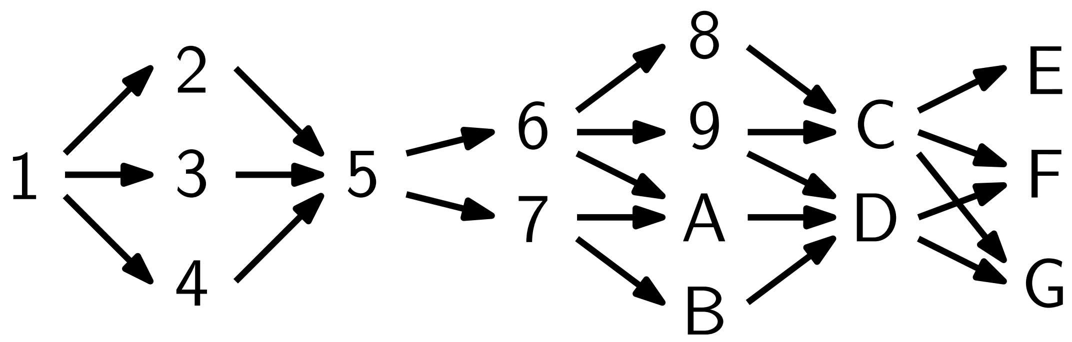
- jobs stored in a list L
  (e.g., topologically sorted)
- a job in L is *available* when all its predecessors have been scheduled
- for each time $t = 1, 2, \dots$ we can schedule $\leq W$ available jobs

Approximating PCMPS

- jobs stored in a list L
  (e.g., topologically sorted)
- a job in L is *available* when all its predecessors have been scheduled
- for each time $t = 1, 2, \ldots$ we can schedule $\leq W$ available jobs
- as long as there are free machines and available jobs, take the first available job and assign it to a free machine

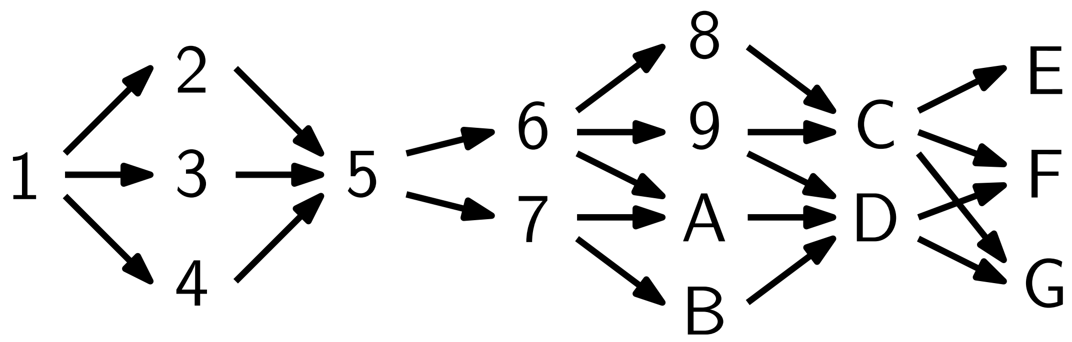
Approximating PCMPS

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



Approximating PCMPS

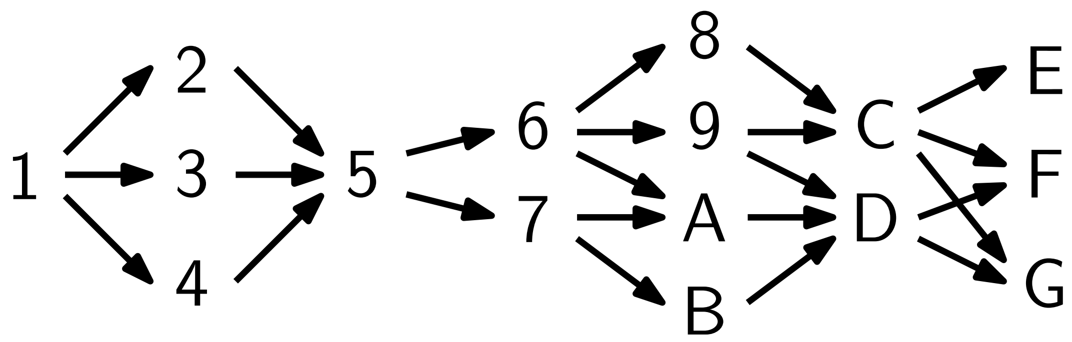
Input: Precedence graph (divided into layers of arbitrary width)



Number of machines is $W = 2$.

Approximating PCMPS

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

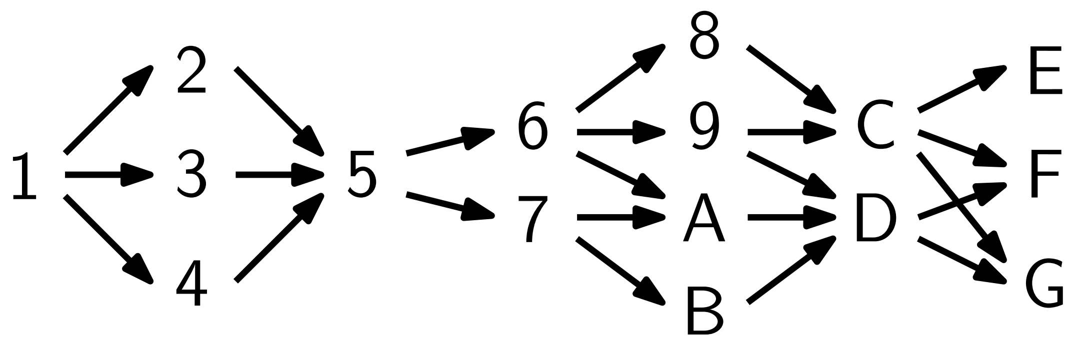


Number of machines is $W = 2$.

Output: Schedule

Approximating PCMPS

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



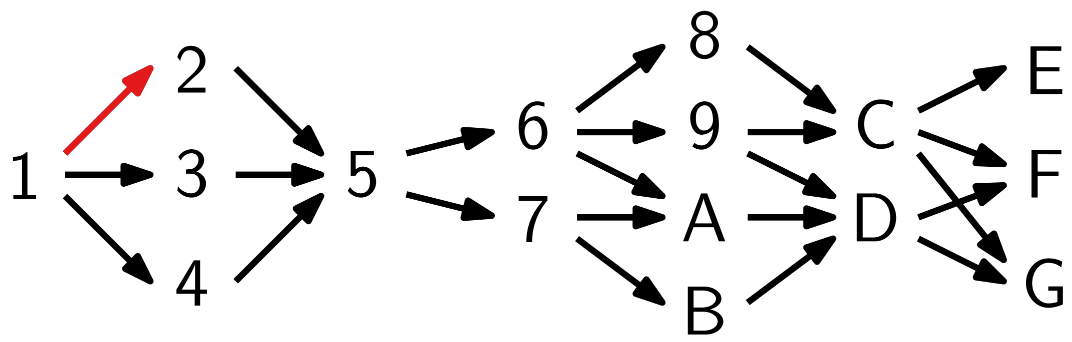
Number of machines is $W = 2$.

Output: Schedule

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

Approximating PCMPS

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



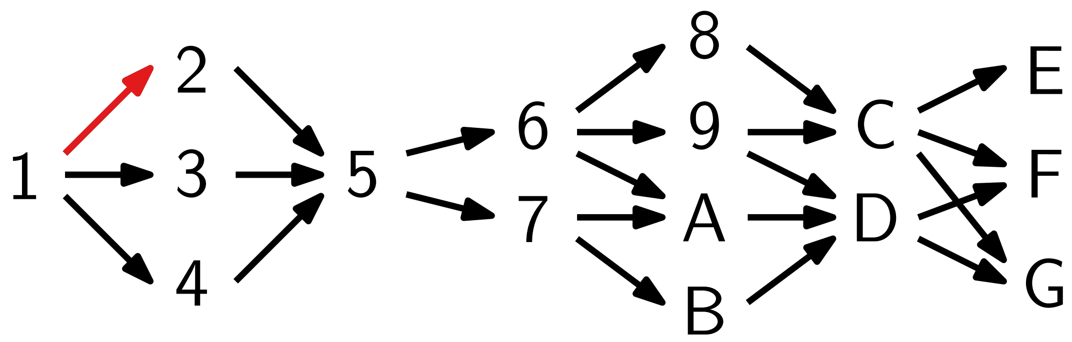
Number of machines is $W = 2$.

Output: Schedule

M_1	1									
M_2	–									
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



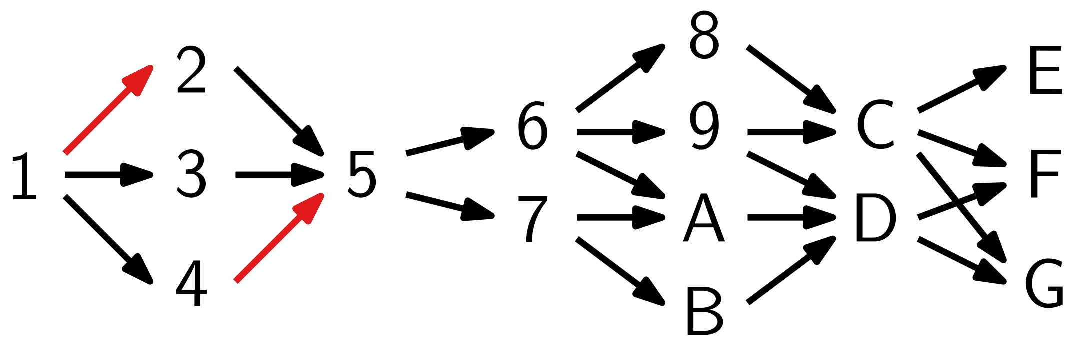
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2								
M_2	–	3								
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



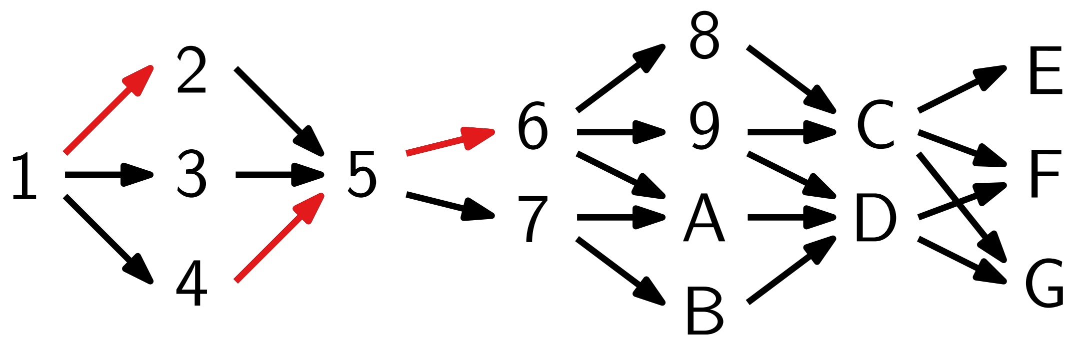
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2	4							
M_2	–	3	–							
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



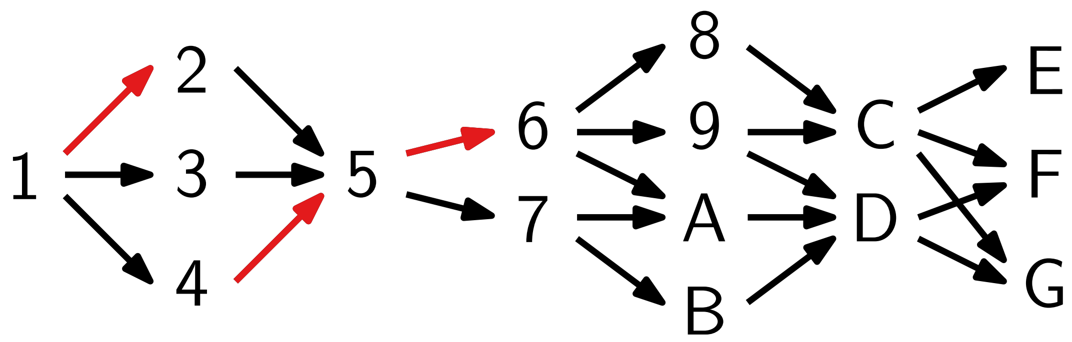
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2	4	5						
M_2	–	3	–	–						
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



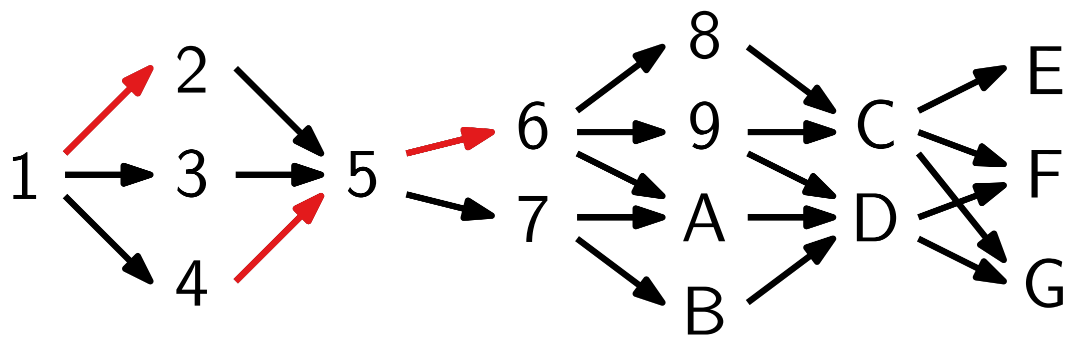
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2	4	5	6					
M_2	–	3	–	–	7					
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



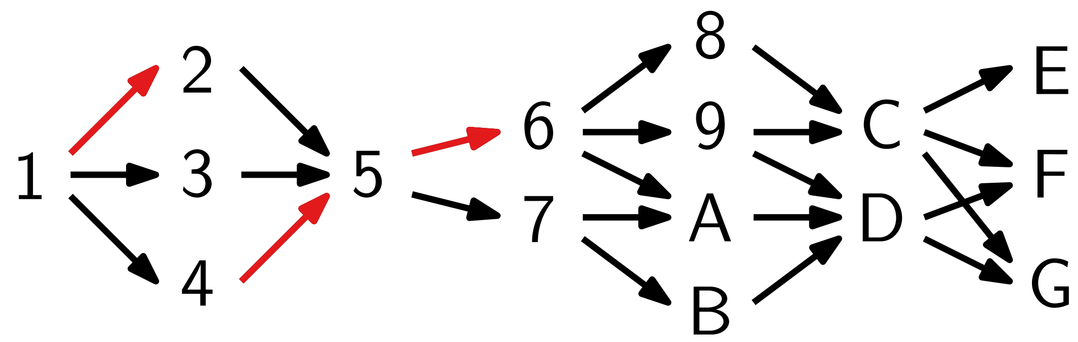
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2	4	5	6	8				
M_2	–	3	–	–	7	9				
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



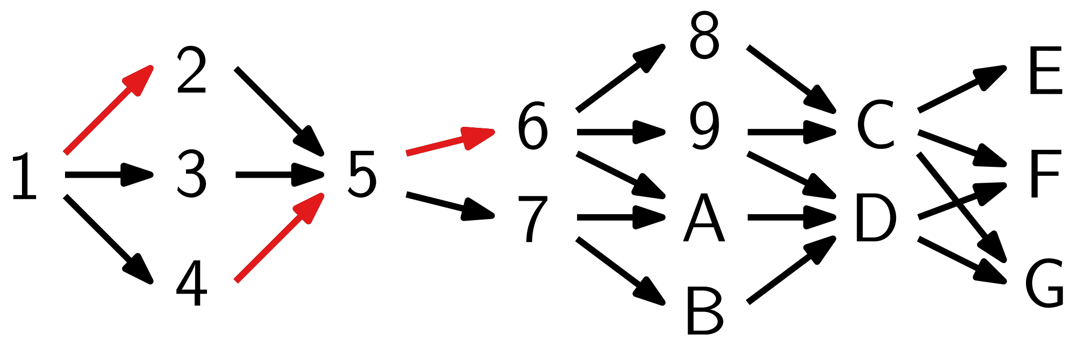
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2	4	5	6	8	A			
M_2	–	3	–	–	7	9	B			
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



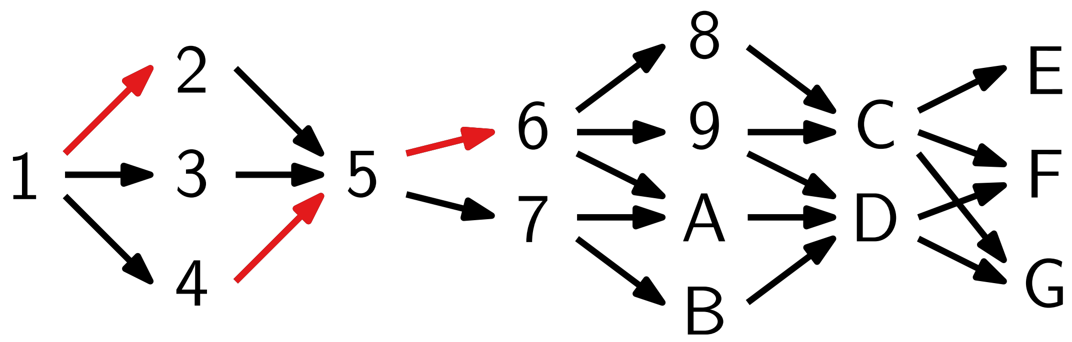
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2	4	5	6	8	A	C		
M_2	–	3	–	–	7	9	B	D		
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



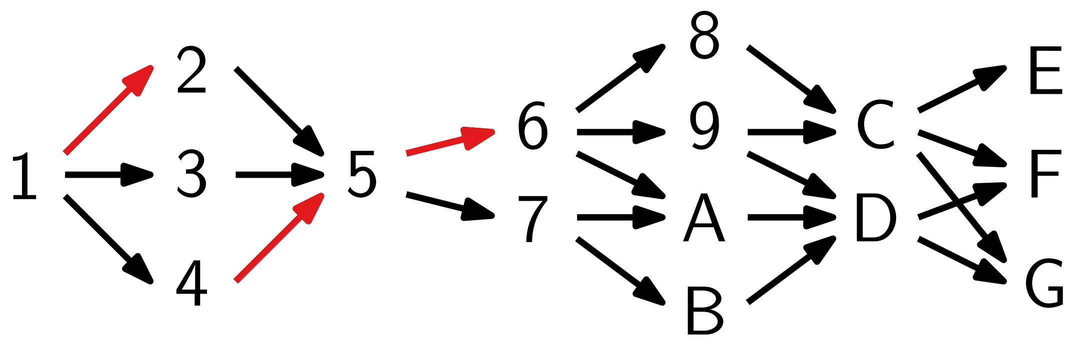
Number of machines is $W = 2$.

Output: Schedule

M_1	1	2	4	5	6	8	A	C	E	
M_2	–	3	–	–	7	9	B	D	F	
t	1	2	3	4	5	6	7	8	9	10

Approximating PCMPS

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



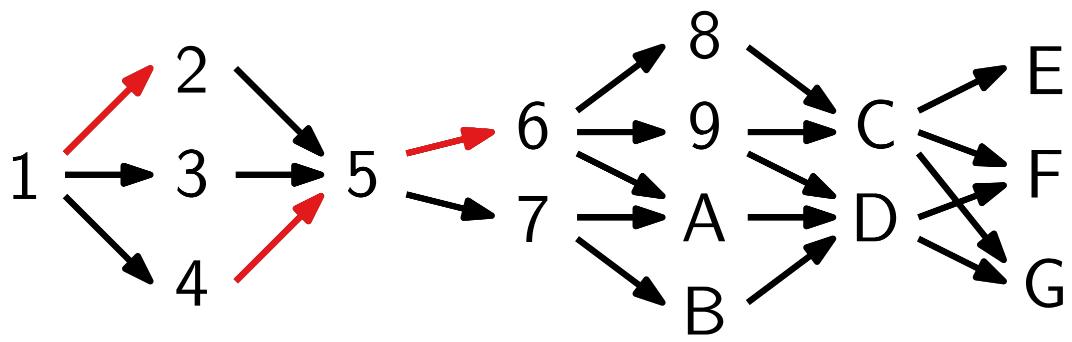
Number of machines is $W = 2$.

Output: 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

Approximating PCMPS

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



Number of machines is $W = 2$.

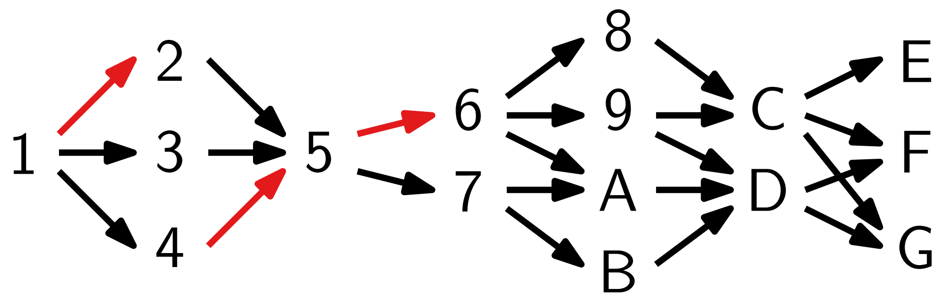
Output: Schedule

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

Question: Good approximation factor?

Approximating PCMPS - Analysis for $W = 2$

Precedence graph $G_<$



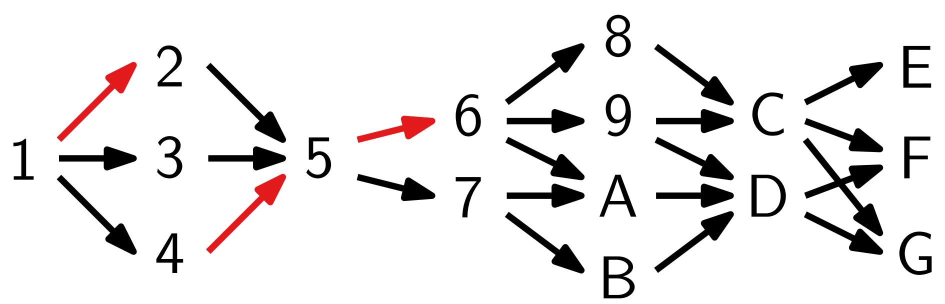
Schedule

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“

Approximating PCMPS - Analysis for $W = 2$

Precedence graph $G_<$



Schedule

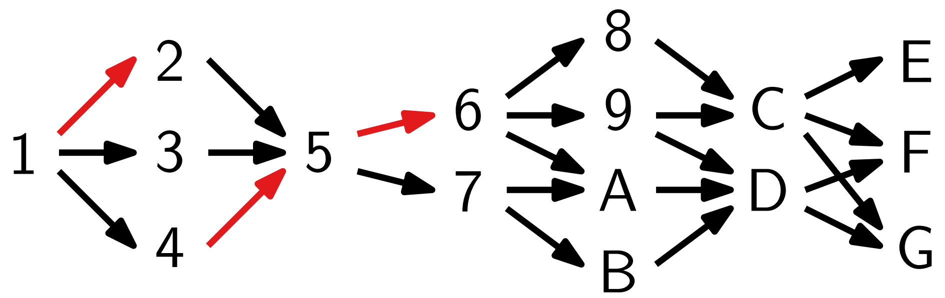
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„The art of the lower bound“

OPT $\geq$

Approximating PCMPS - Analysis for $W = 2$

Precedence graph $G_<$



Schedule

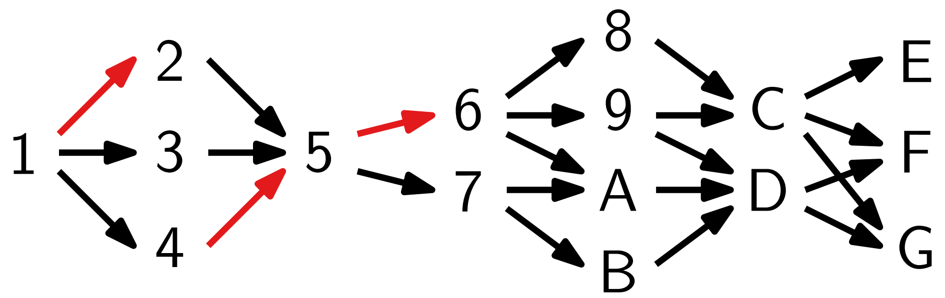
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„The art of the lower bound“

OPT $\geq \lceil n/2 \rceil$

Approximating PCMPS - Analysis for $W = 2$

Precedence graph $G_<$



Schedule

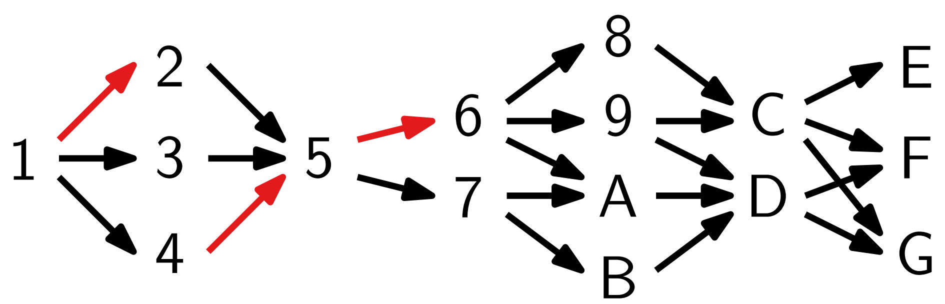
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„The art of the lower bound“

OPT $\geq$ $\lceil n/2 \rceil$ and OPT $\geq$

Approximating PCMPS - Analysis for $W = 2$

Precedence graph $G_<$



Schedule

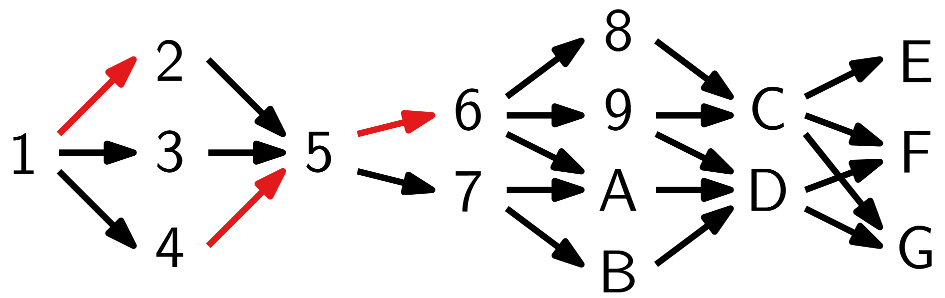
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„The art of the lower bound“

OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

Approximating PCMPS - 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

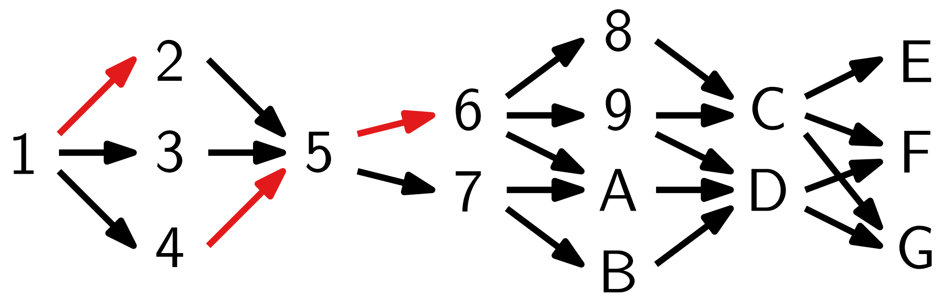
„The art of the lower bound"

OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

Goal: measure the quality of our algorithm using the lower bounds

Approximating PCMPS - 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"

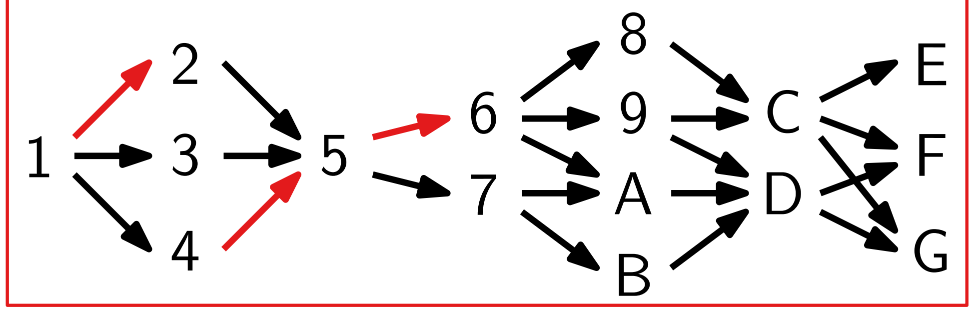
OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

Goal: measure the quality of our algorithm using the lower bounds

Bound. ALG $\leq$

Approximating PCMPS - 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“

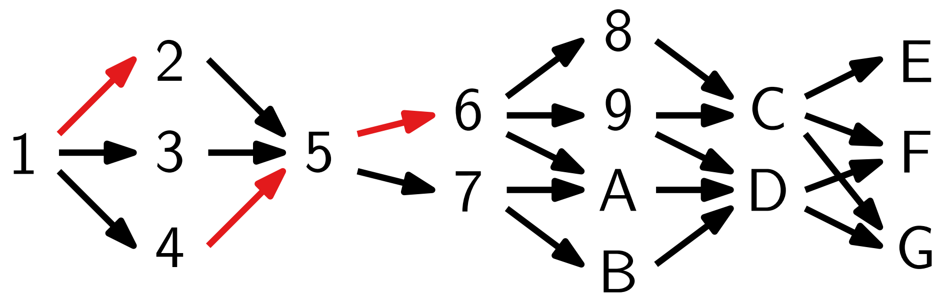
OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

Goal: measure the quality of our algorithm using the lower bounds

Bound. ALG $\leq$

Approximating PCMPS - 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 $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

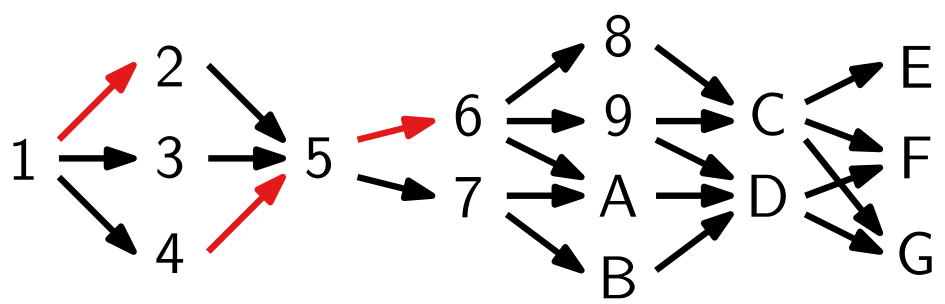
Goal: measure the quality of our algorithm using the lower bounds

Bound. ALG $\leq$

insertion of pauses (–) in the schedule
(except the last) maps to layers of $G_<$

Approximating PCMPS - 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 $\geq \ell :=$ Number of layers of $G_<$ (= length of longest path in $G_<$)

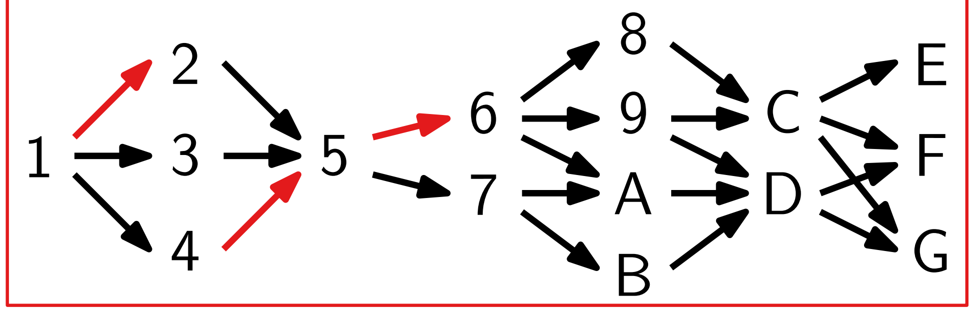
Goal: measure the quality of our algorithm using the lower bounds

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

insertion of pauses (–) in the schedule
(except the last) maps to layers of $G_<$

Approximating PCMPS - 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 $\geq$ $\ell :=$ Number of layers of $G_<$ (= length of longest path in $G_<$)

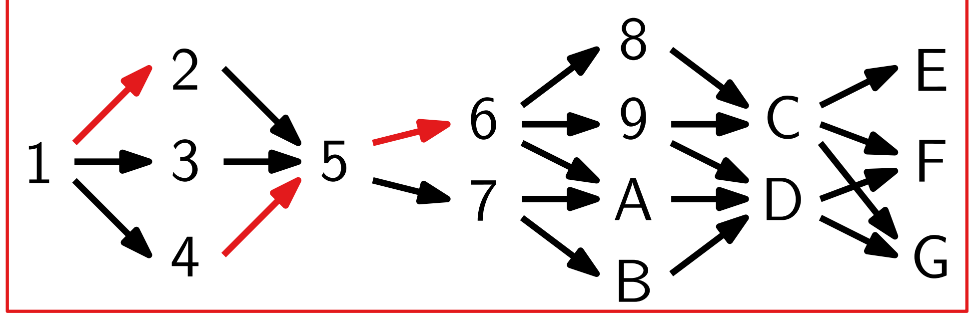
Goal: measure the quality of our algorithm using the lower bounds

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

insertion of pauses (–) in the schedule
(except the last) maps to layers of $G_<$

Approximating PCMPS - 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 $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

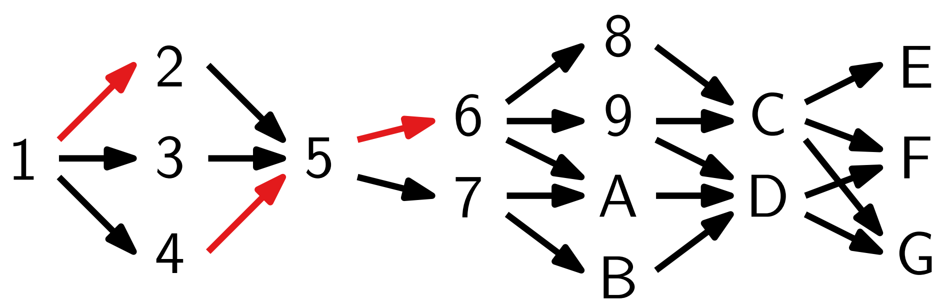
Goal: measure the quality of our algorithm using the lower bounds

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

insertion of pauses (–) in the schedule
(except the last) maps to layers of $G_<$

Approximating PCMPS - 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 $\geq \ell :=$ Number of layers of $G_<$ (= length of longest path in $G_<$)

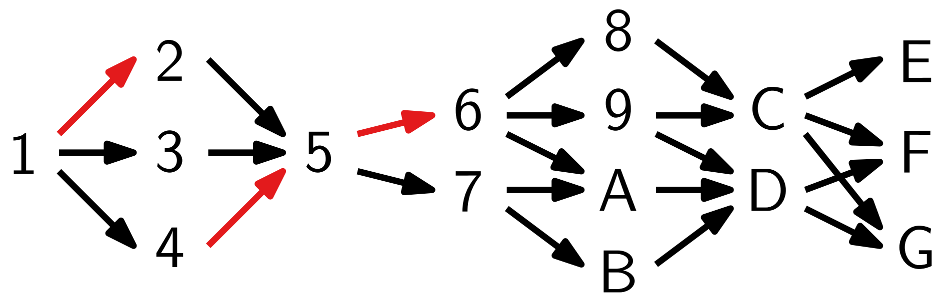
Goal: measure the quality of our algorithm using the lower bounds

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

insertion of pauses (–) in the schedule
(except the last) maps to layers of $G_<$

Approximating PCMPS - 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 $\geq \ell :=$ Number of layers of $G_<$ (= length of longest path in $G_<$)

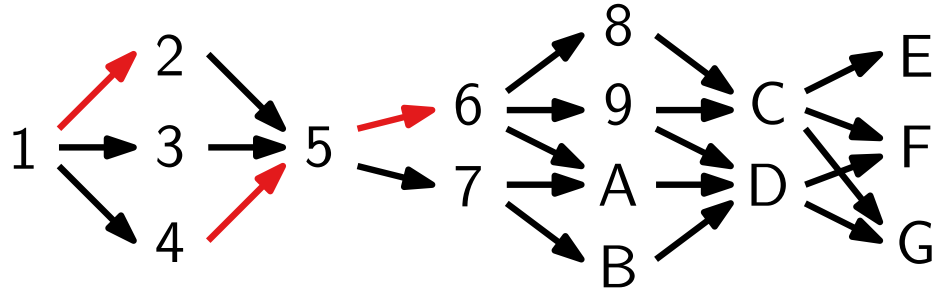
Goal: measure the quality of our algorithm using the lower bounds

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

insertion of pauses (–) in the schedule
(except the last) maps to layers of $G_<$

Approximating PCMPS - 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 $\geq \ell :=$ Number of layers of $G_<$ (= length of longest path in $G_<$)

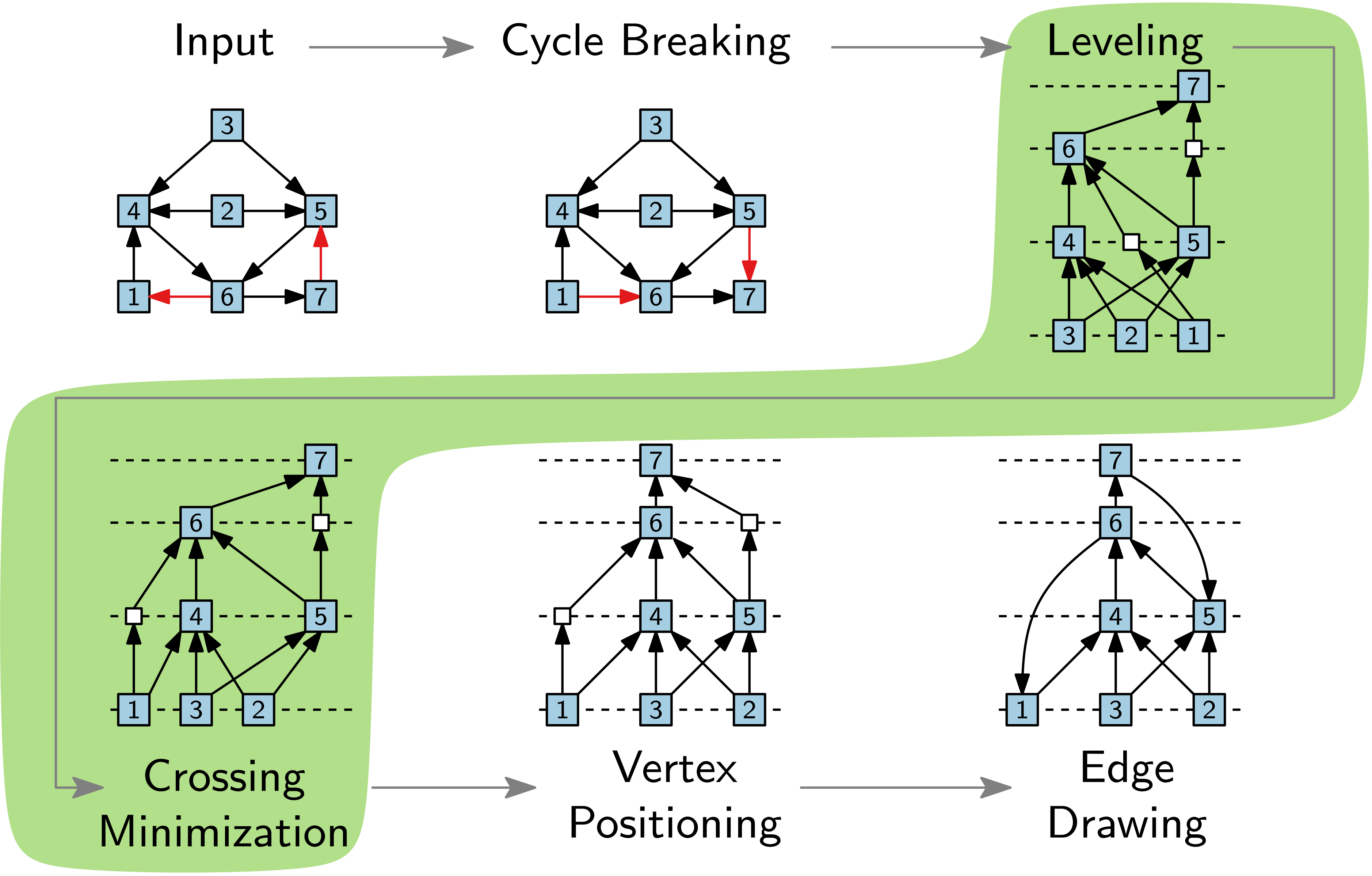
Goal: measure the quality of our algorithm using the lower bounds

$\leq (2 - 1/W) \cdot$ OPT in general case

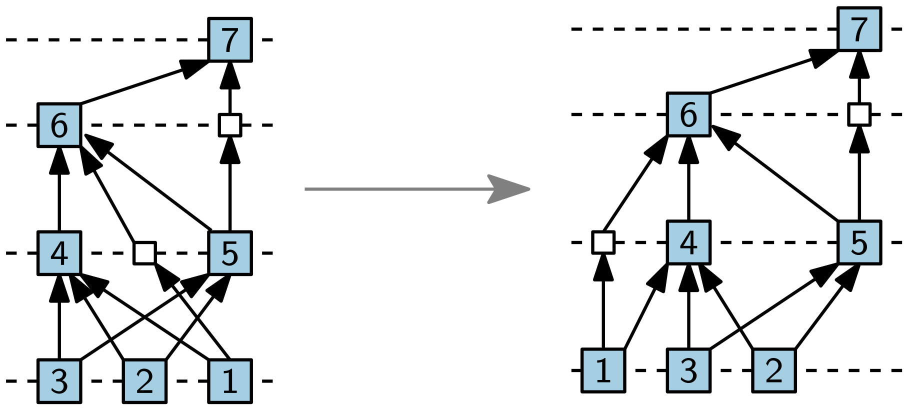
Bound. ALG $\leq \left\lceil \frac{n+\ell}{2} \right\rceil \approx \lceil n/2 \rceil + \ell/2 \leq 3/2 \cdot$ OPT

insertion of pauses (–) in the schedule (except the last) maps to layers of $G_<$

Step 3: Crossing Minimization

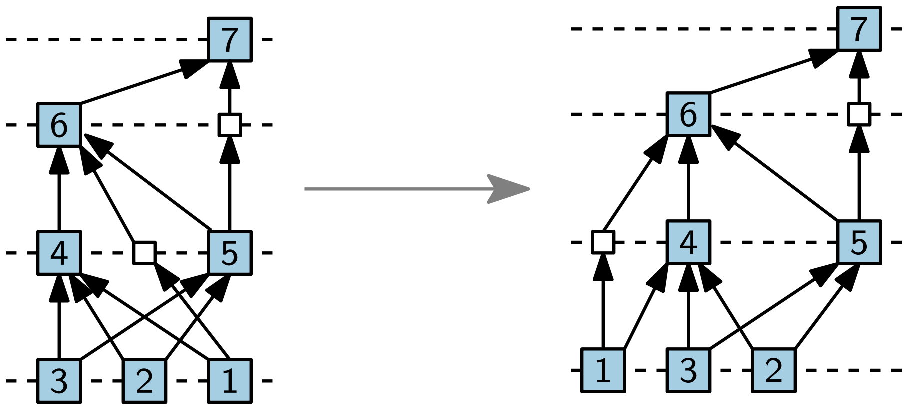


Step 3: Crossing Minimization



Problem.

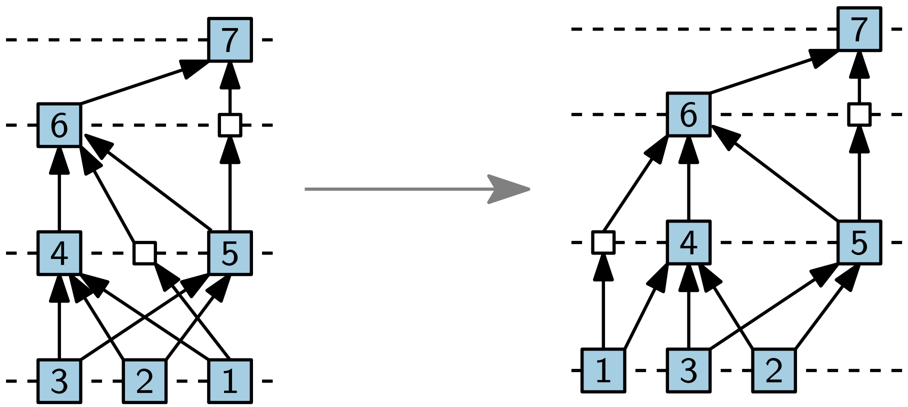
Step 3: Crossing Minimization



Problem.

- Input: Graph G, leveling $y\colon V \to \{1, \ldots, n\}$

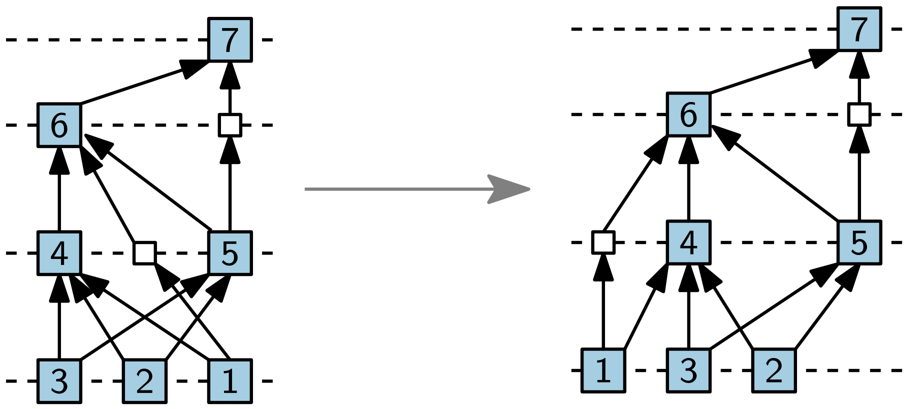
Step 3: Crossing Minimization



Problem.

- Input: Graph G, leveling $y\colon V \to \{1, \ldots, n\}$
- Output: (Re-)ordering of vertices in each layer such that the number of crossings is minimized.

Step 3: Crossing Minimization

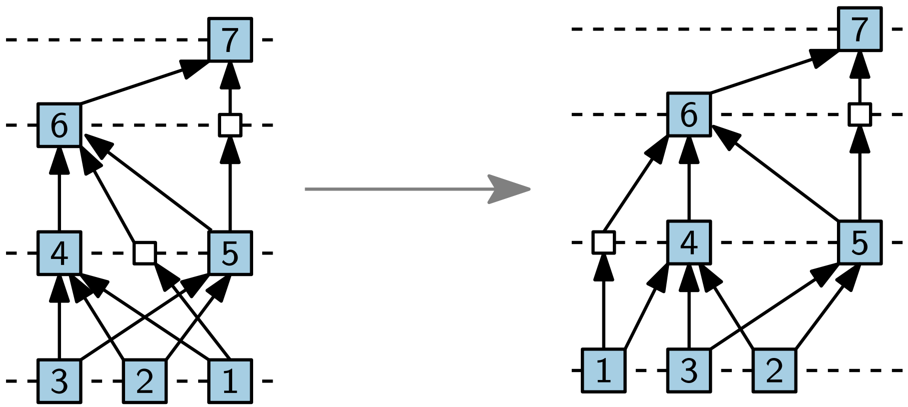


Problem.

- Input: Graph G, leveling $y\colon V \to \{1, \ldots, n\}$
- Output: (Re-)ordering of vertices in each layer such that the number of crossings is minimized.

- NP-hard, even for 2 layers [Garey & Johnson '83]

Step 3: Crossing Minimization



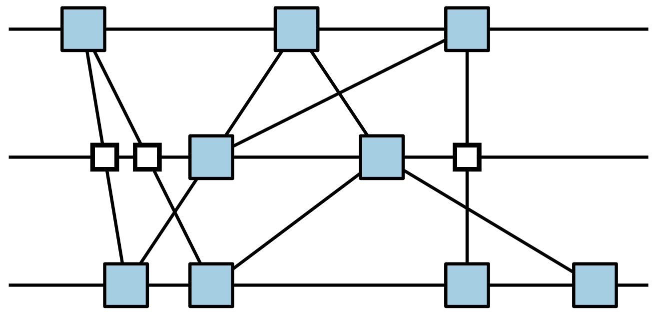
Problem.

- Input: Graph G, leveling $y\colon V \to \{1, \ldots, n\}$
- Output: (Re-)ordering of vertices in each layer such that the number of crossings is minimized.

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

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

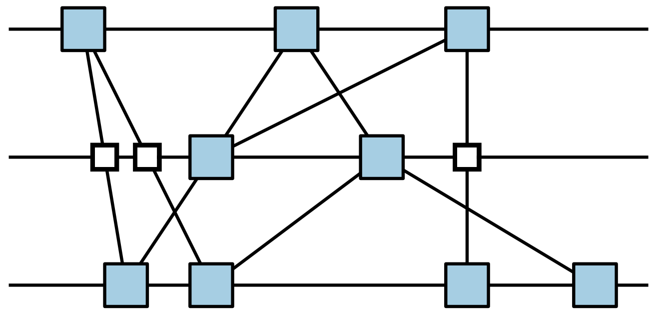


Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like “regular” vertices

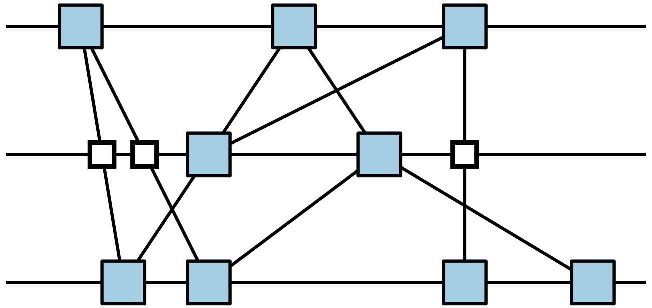


Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like “regular” vertices



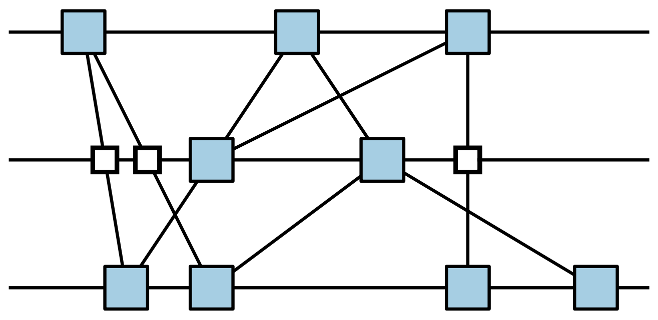
Algorithm scheme.

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like “regular” vertices



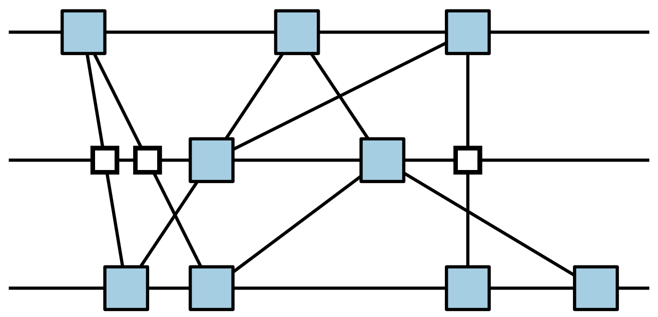
Algorithm scheme.

(1) choose a random permutation of L_1

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.



- permute one layer after the other
- treat dummy-vertices like "regular" vertices

Algorithm scheme.

(1) choose a random permutation of L_1

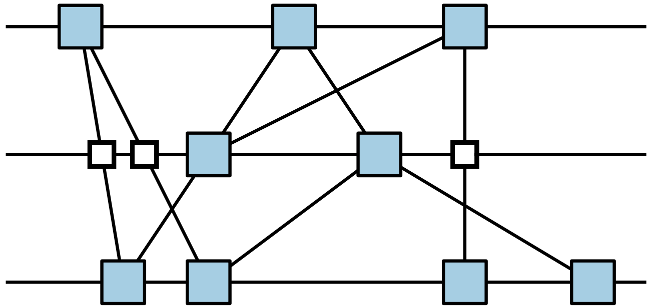
(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like “regular” vertices



Algorithm scheme.

(1) choose a random permutation of L_1

(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

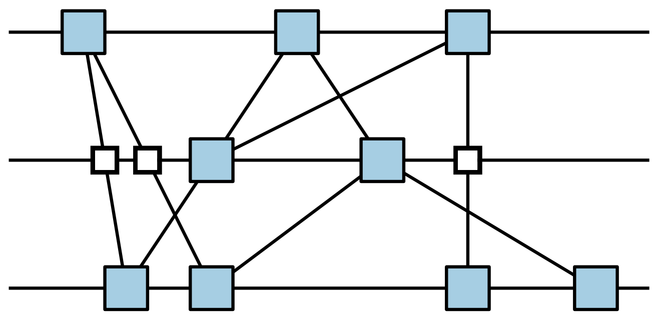
(3) minimize crossings by permuting L_{i+1} while keeping L_i fixed

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like "regular" vertices



Algorithm scheme.

(1) choose a random permutation of L_1

(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

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

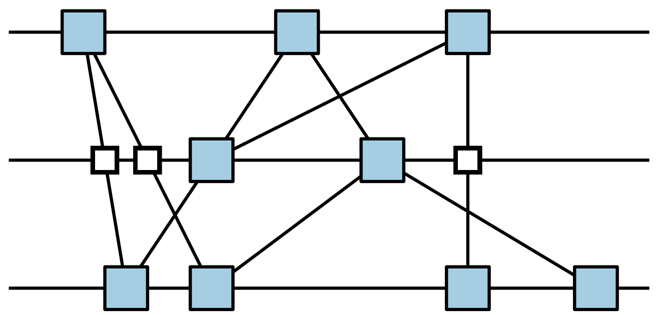
(4) repeat steps (2)–(3) in the reverse order (starting from topmost layer L_h)

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like “regular” vertices



Algorithm scheme.

(1) choose a random permutation of L_1

(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

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

(4) repeat steps (2)–(3) in the reverse order (starting from topmost layer L_h)

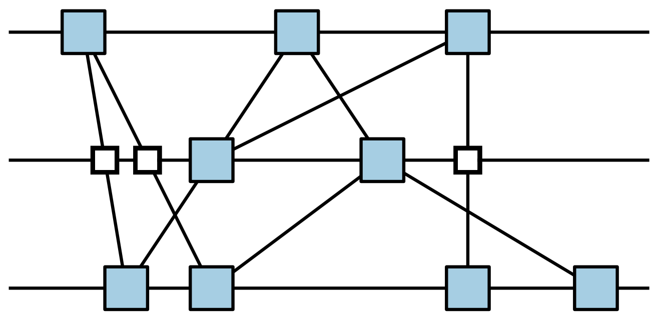
(5) repeat steps (2)–(4) until no further improvement is achieved

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like “regular” vertices



Algorithm scheme.

(1) choose a random permutation of L_1

(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

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

(4) repeat steps (2)–(3) in the reverse order (starting from topmost layer L_h)

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

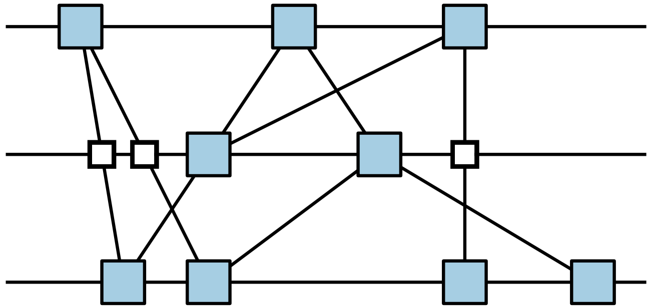
(6) repeat steps (1)–(5) with different starting permutations on L_1

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like "regular" vertices



Algorithm scheme.

(1) choose a random permutation of L_1

(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

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

(4) repeat steps (2)–(3) in the reverse order (starting from topmost layer L_h)

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

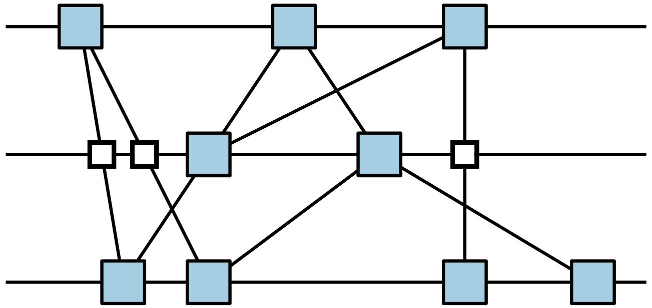
(6) repeat steps (1)–(5) with different starting permutations on L_1

Iterative Crossing Reduction

Observation. The number of crossings only depends on permutations of adjacent layers.

Idea.

- permute one layer after the other
- treat dummy-vertices like “regular” vertices



Algorithm scheme.

(1) choose a random permutation of L_1

one-sided crossing minimization

(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

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

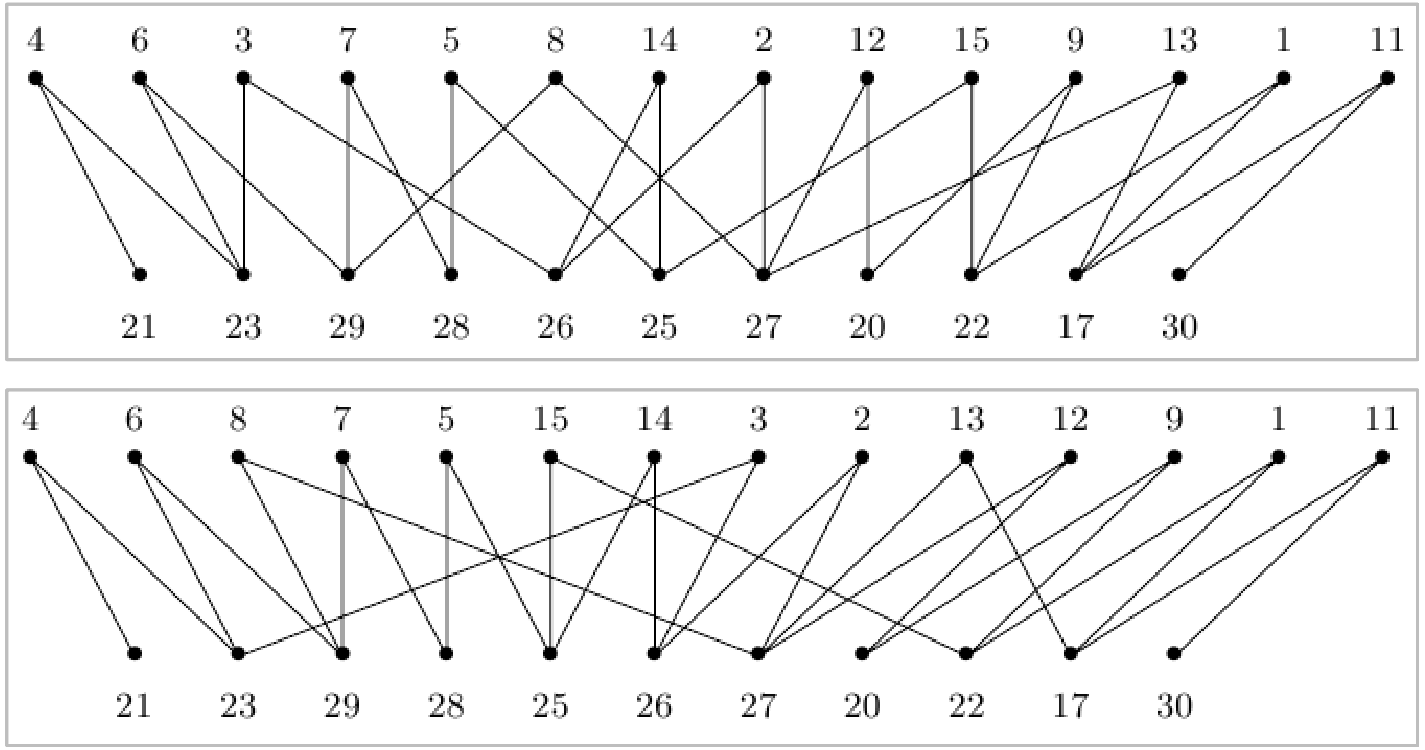
(4) repeat steps (2)–(3) in the reverse order (starting from topmost layer L_h)

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

(6) repeat steps (1)–(5) with different starting permutations on L_1

One-Sided Crossing Minimization

Problem.

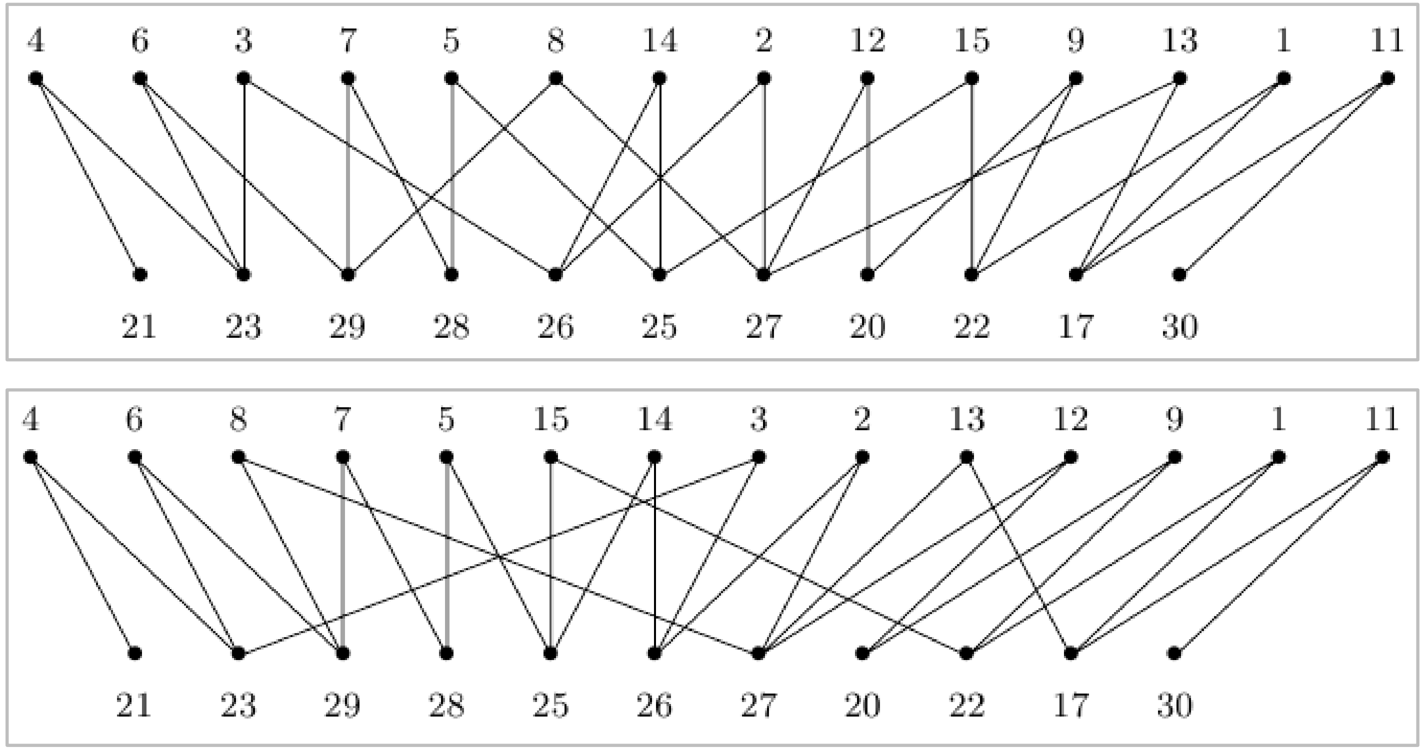


[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1

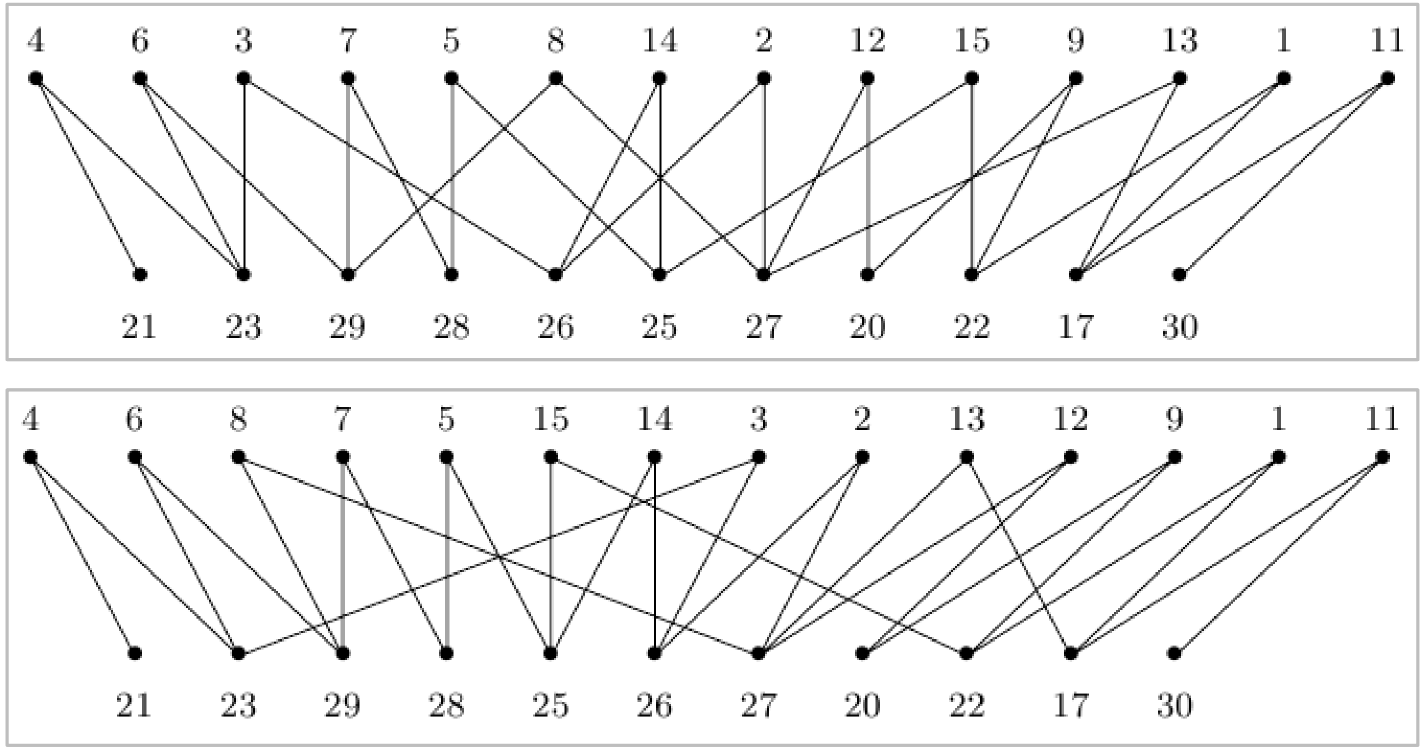


[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.



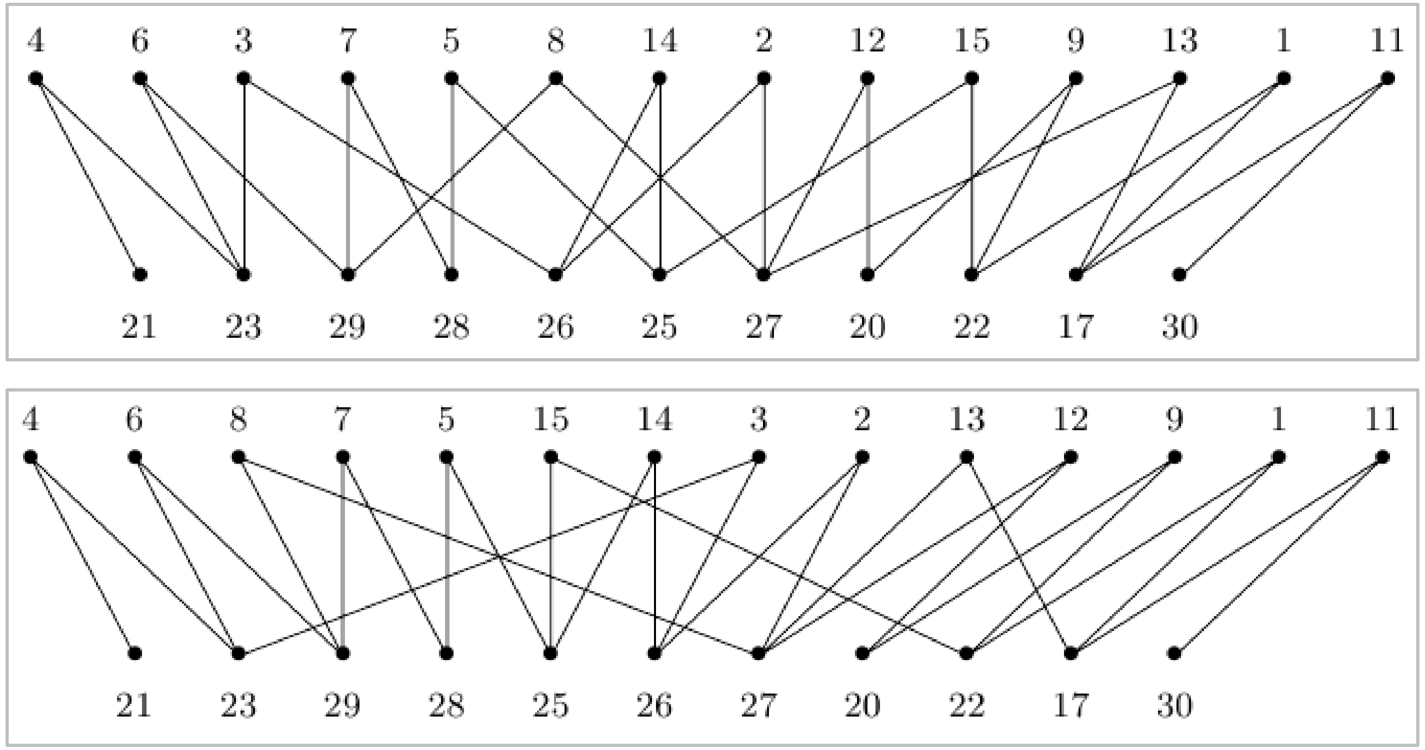
[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.
[Eades & Whitesides '94]



[Kaufmann & Wagner: Drawing Graphs]

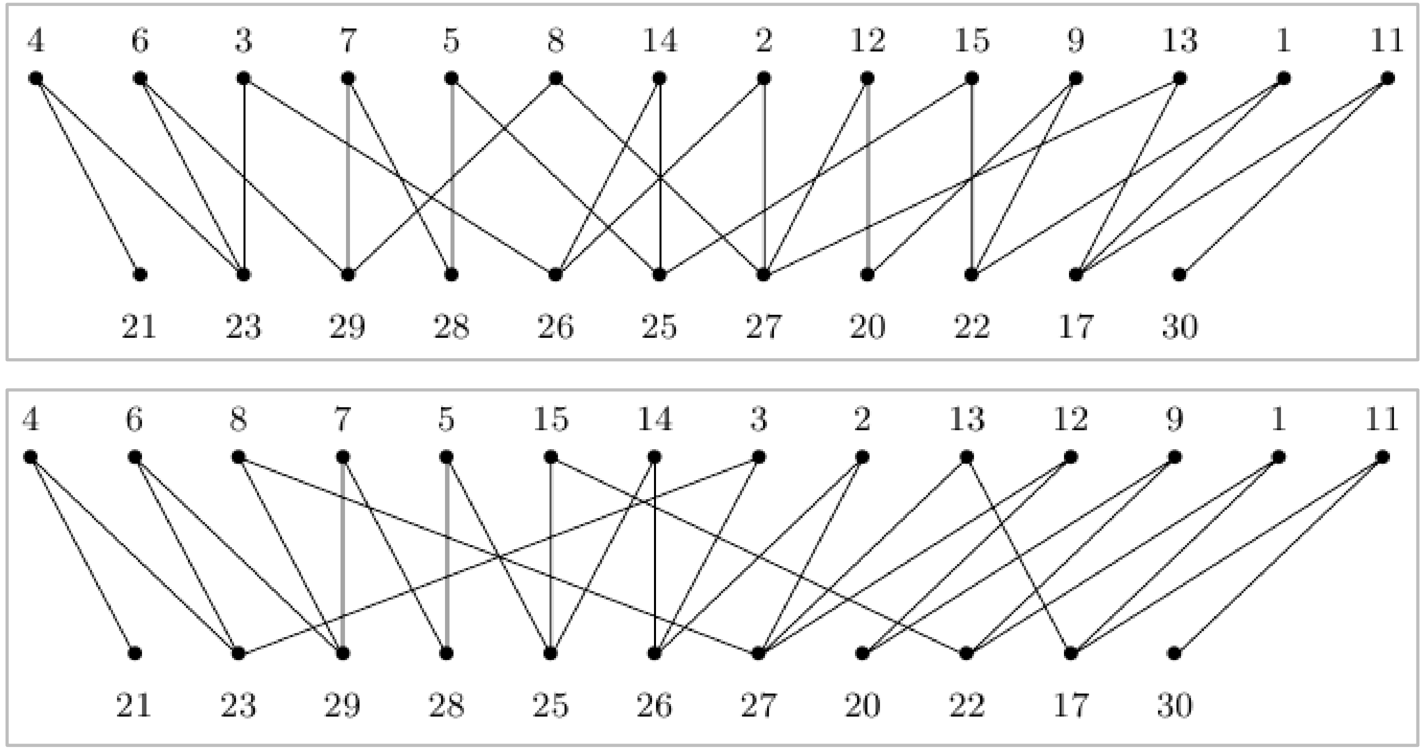
One-Sided Crossing Minimization

Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.
[Eades & Whitesides '94]

Algorithms.



[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

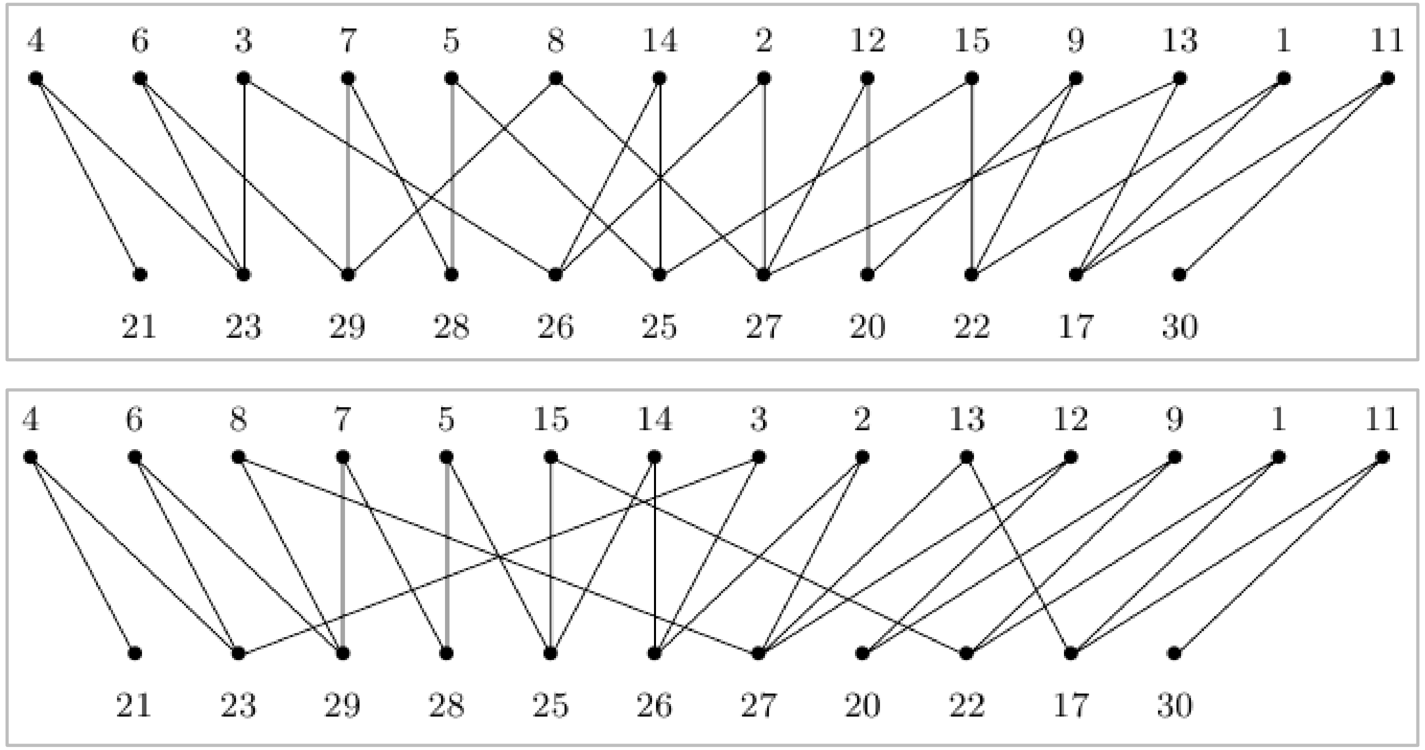
Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.
[Eades & Whitesides '94]

Algorithms.

- barycenter heuristic



[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

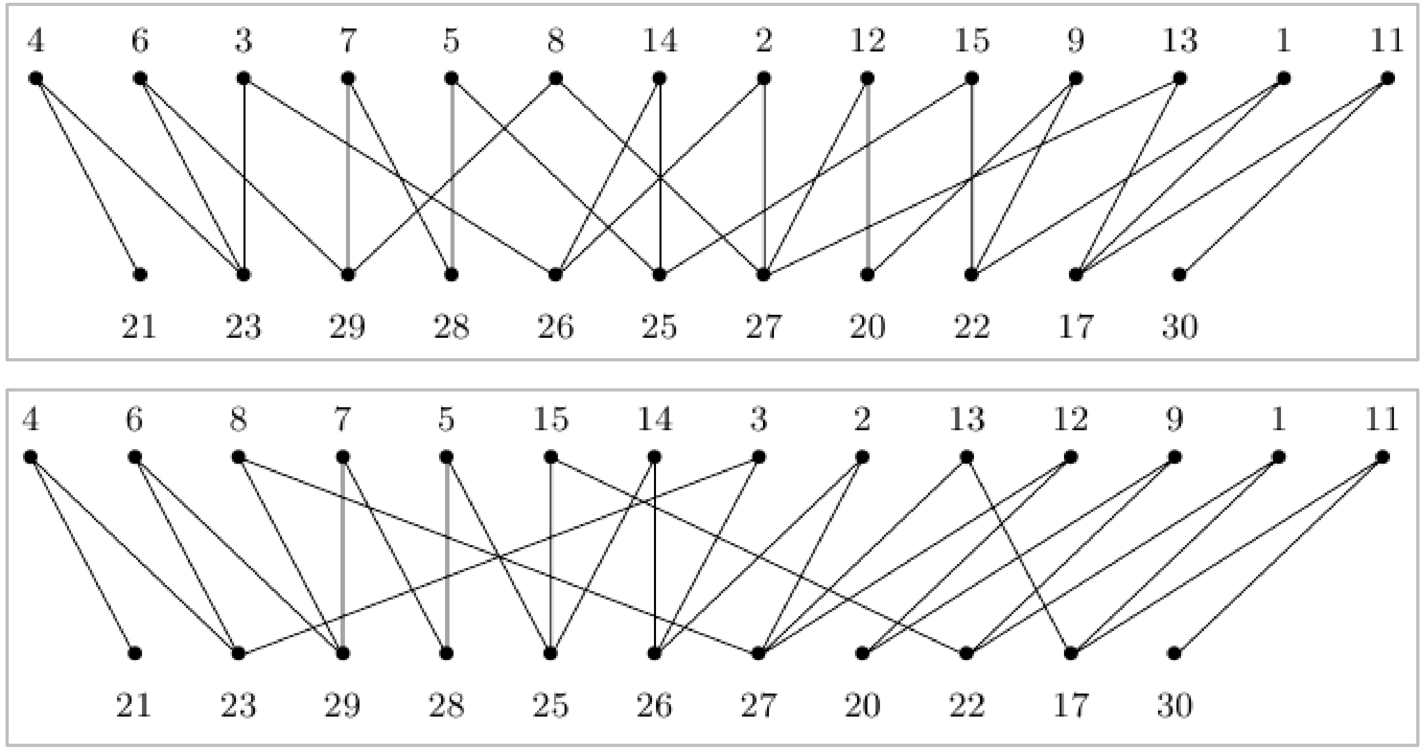
Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.
[Eades & Whitesides '94]

Algorithms.

- barycenter heuristic
- median heuristic



[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

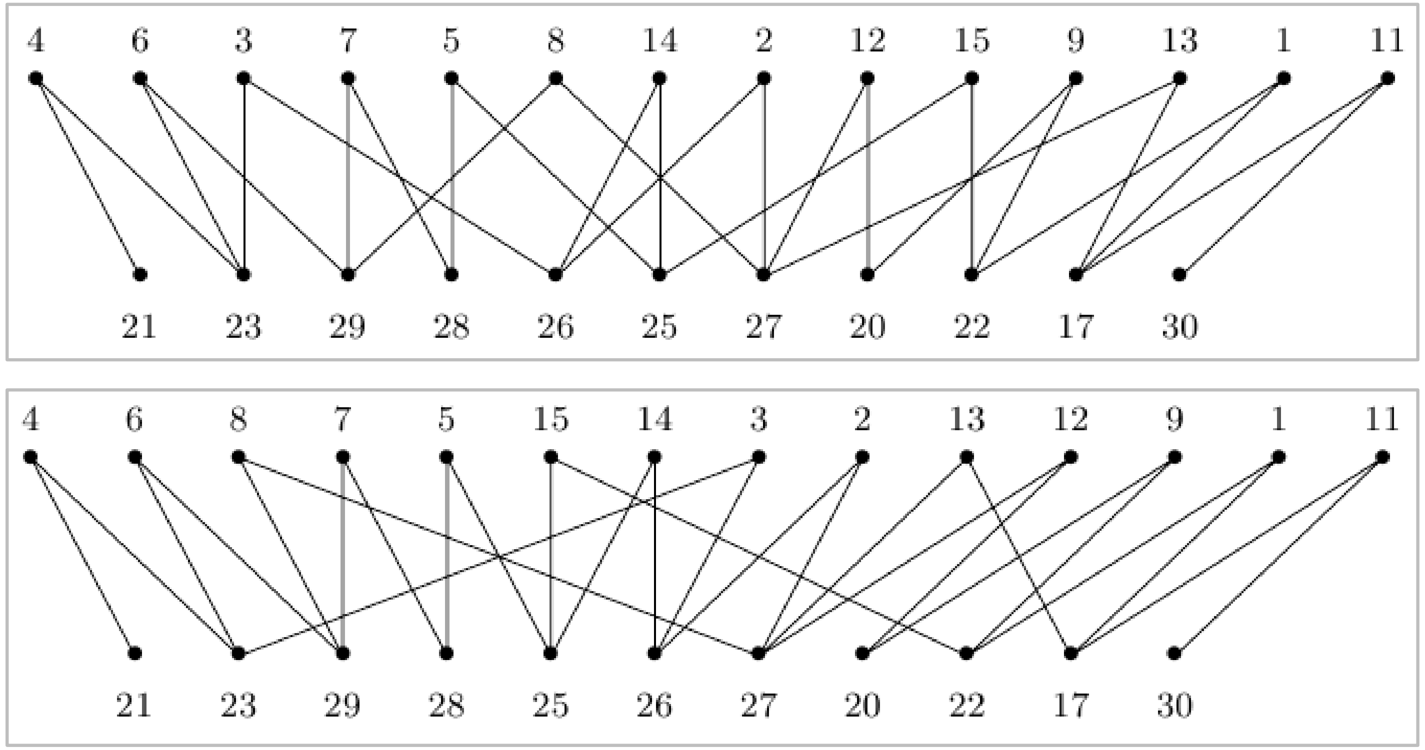
Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.
[Eades & Whitesides '94]

Algorithms.

- barycenter heuristic
- median heuristic
- Greedy-Switch



[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

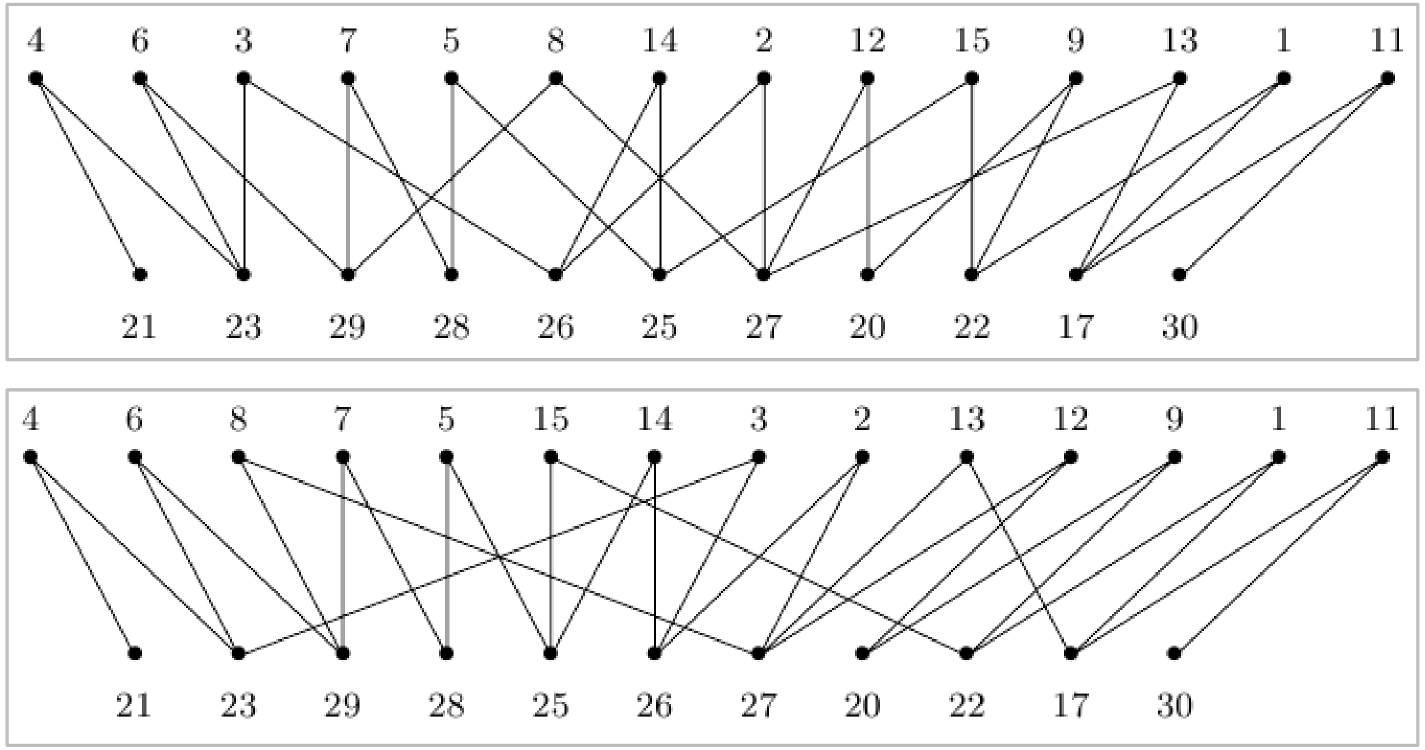
Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.
[Eades & Whitesides '94]

Algorithms.

- barycenter heuristic
- median heuristic
- Greedy-Switch
- ILP



[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

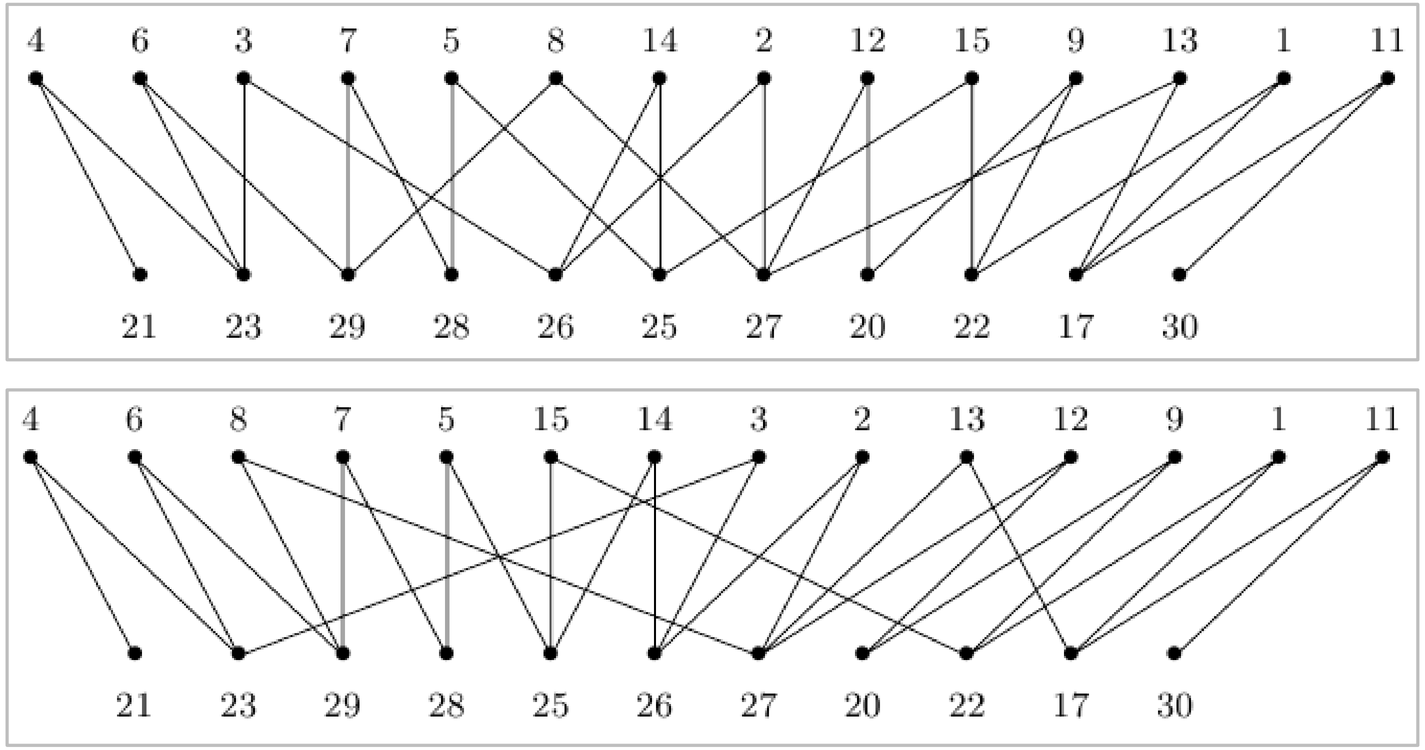
Problem.

- Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.
[Eades & Whitesides '94]

Algorithms.

- barycenter heuristic
- median heuristic
- Greedy-Switch
- ILP
- ...



[Kaufmann & Wagner: Drawing Graphs]

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors
- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\mathrm{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\mathrm{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{bary}(u)$.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\mathrm{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{bary}(u)$.
- vertices with the same barycenter keep their old relative ranks

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\text{bary}(u)$.
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\text{bary}(u)$.
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)
- relatively good results

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\mathrm{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, bary(u).
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\mathrm{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{bary}(u)$.
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required
- $O(\sqrt{n})$-approximation factor

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\text{bary}(u)$.
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- $O(\sqrt{n})$-approximation factor

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors
- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

Worst case?

$$\mathrm{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{bary}(u)$.
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- $O(\sqrt{n})$-approximation factor

Barycenter Heuristic

[Sugiyama et al. '81]

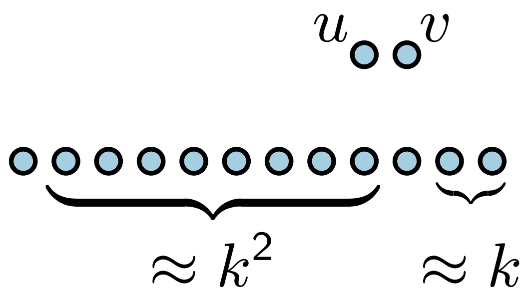
- Intuition: few intersections occur when vertices are close to their neighbors

- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\text{bary}(u)$.
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- $O(\sqrt{n})$-approximation factor

Worst case?



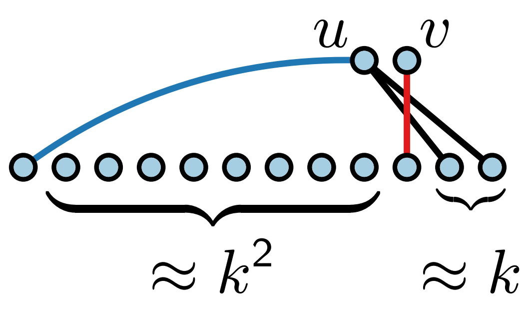
Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbors
- The barycenter of $u \in L_2$ is the mean rank of u's neighbors on layer L_1.

  $\text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} \pi_1(v)$

Worst case?



- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, bary(u).
- vertices with the same barycenter keep their old relative ranks
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- $O(\sqrt{n})$-approximation factor

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

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

- $\mathrm{med}(u) :=$

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ & \end{cases}$$

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, med(u).

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, med(u).
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\mathrm{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{med}(u)$.
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\mathrm{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{med}(u)$.
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)
- relatively good results

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, med(u).
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\mathrm{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{med}(u)$.
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\text{med}(u)$.
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- 3-approximation factor

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, med(u).
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- 3-approximation factor

  Proof in [GD Ch 11]

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, med(u).
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)

Worst case?

- relatively good results
- optimal if no crossings are required ← Exercise!
- 3-approximation factor

Proof in [GD Ch 11]

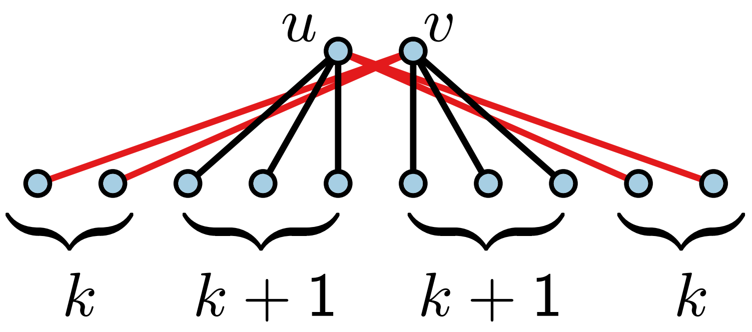
Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\text{med}(u)$.
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- 3-approximation factor

  Proof in [GD Ch 11]

Worst case?



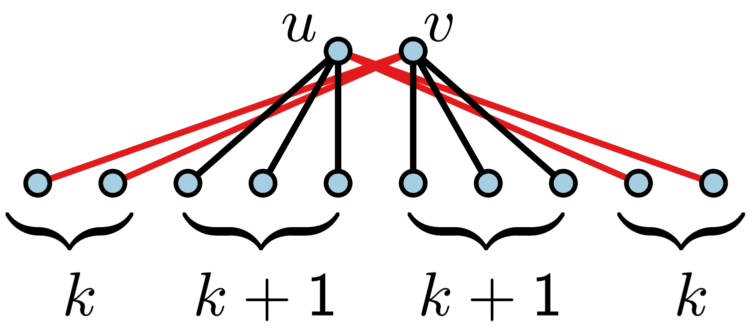
Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\mathrm{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise} \end{cases}$$
- To get π_2, sort L_2 ascendingly using, for each $u \in L_2$, $\mathrm{med}(u)$.
- for vertices with the same median, we place vertices with odd degree to the left of vertices with even degree (and keep their old relative ranks among the odd/even vertices)
- linear runtime (in the number of vertices and edges)
- relatively good results
- optimal if no crossings are required ← Exercise!
- 3-approximation factor

  Proof in [GD Ch 11]

Worst case?



crossings: $2k(k+1) + k^2$ vs. $(k+1)^2$

Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.

Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- runtime $O(|L_2|)$ per iteration; at most $|L_2|$ iterations $\Rightarrow O(|L_2|^2)$

Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- runtime $O(|L_2|)$ per iteration; at most $|L_2|$ iterations $\Rightarrow O(|L_2|^2)$
- suitable as post-processing for other heuristics

Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- runtime $O(|L_2|)$ per iteration; at most $|L_2|$ iterations $\Rightarrow O(|L_2|^2)$
- suitable as post-processing for other heuristics

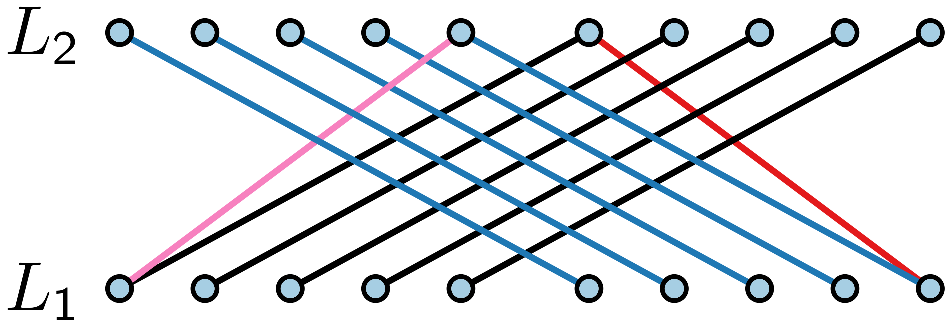
Worst case?

Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- runtime $O(|L_2|)$ per iteration; at most $|L_2|$ iterations $\Rightarrow O(|L_2|^2)$
- suitable as post-processing for other heuristics

Worst case?

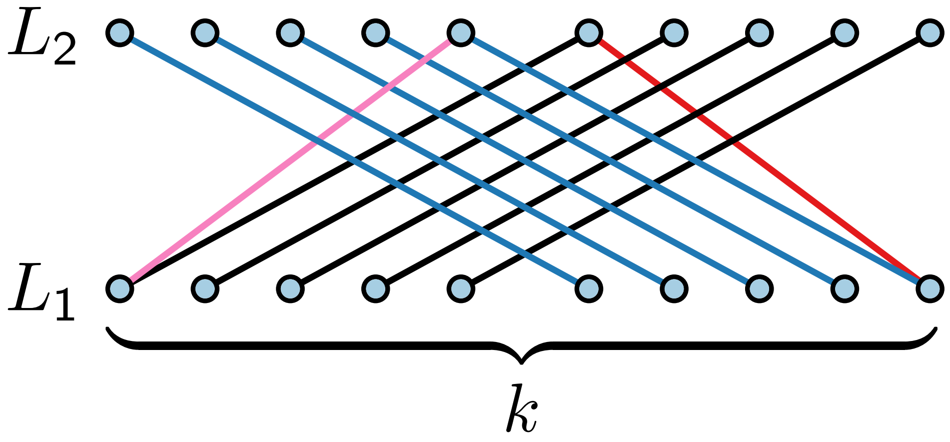


Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- runtime $O(|L_2|)$ per iteration; at most $|L_2|$ iterations $\Rightarrow O(|L_2|^2)$
- suitable as post-processing for other heuristics

Worst case?

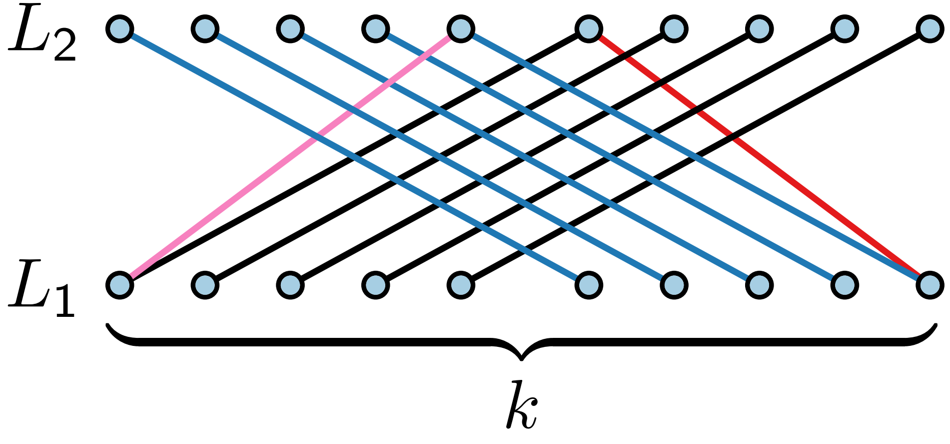


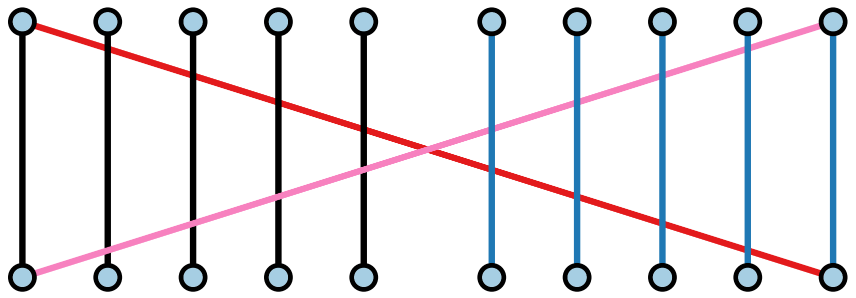
Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- runtime $O(|L_2|)$ per iteration; at most $|L_2|$ iterations $\Rightarrow O(|L_2|^2)$
- suitable as post-processing for other heuristics

Worst case?



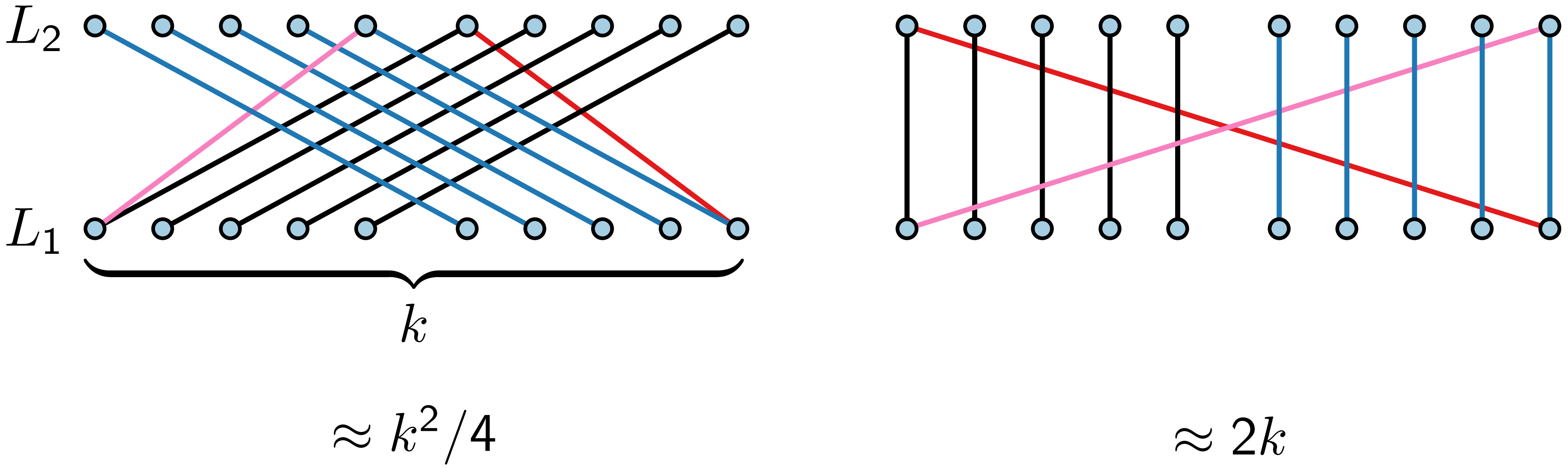


Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- runtime $O(|L_2|)$ per iteration; at most $|L_2|$ iterations $\Rightarrow O(|L_2|^2)$
- suitable as post-processing for other heuristics

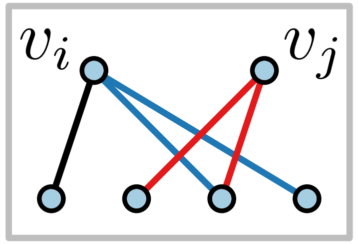
Worst case?



Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

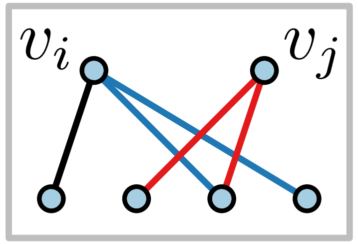


$c_{ij} = 3$

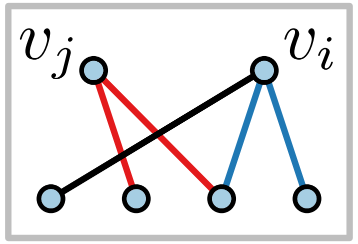
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$



$c_{ij} = 3$



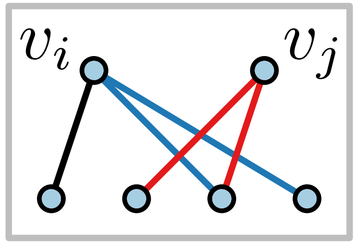
$c_{ji} = 2$

Integer Linear Program (ILP)

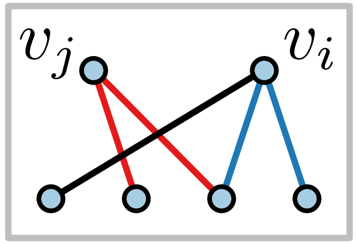
[Jünger & Mutzel, ’97]

- constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

- variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$



$c_{ij} = 3$



$c_{ji} = 2$

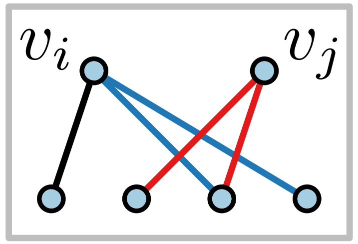
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

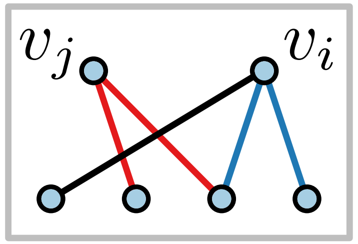
- constant $c_{ij} := \#$ crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

- variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$

$$x_{ij} = \begin{cases} 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\ 0 & \text{otherwise} \end{cases}$$



$c_{ij} = 3$



$c_{ji} = 2$

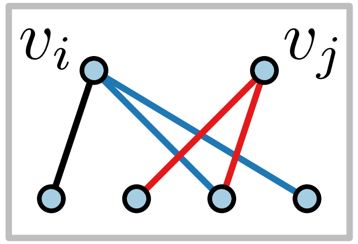
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

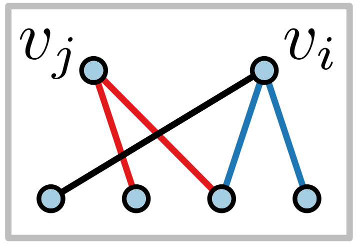
- constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

- variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$

$$x_{ij} = \begin{cases} 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\ 0 & \text{otherwise} \end{cases}$$



$c_{ij} = 3$



$c_{ji} = 2$

- number of crossings of a permutations π_2:

$$\text{cross}(\pi_2) =$$

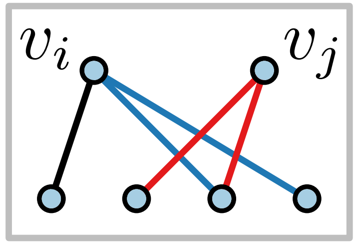
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

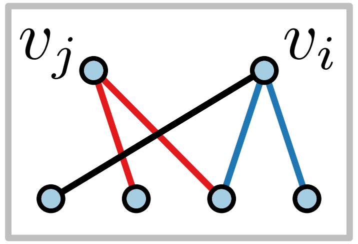
- constant $c_{ij} := \#$ crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

- variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$

$$x_{ij} = \begin{cases} 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\ 0 & \text{otherwise} \end{cases}$$



$c_{ij} = 3$



$c_{ji} = 2$

- number of crossings of a permutations π_2:

$$\text{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2}$$

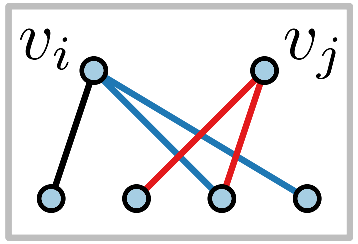
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

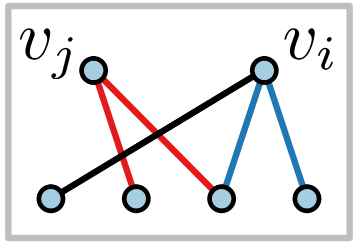
- constant $c_{ij} := \#$ crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

- variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$

$$x_{ij} = \begin{cases} 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\ 0 & \text{otherwise} \end{cases}$$



$c_{ij} = 3$



$c_{ji} = 2$

- number of crossings of a permutations π_2:

$$\mathrm{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji}) x_{ij}$$

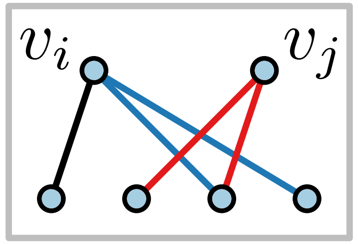
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

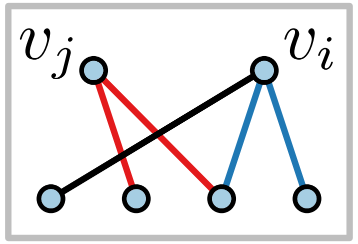
- constant $c_{ij} := \#$ crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

- variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$

$$x_{ij} = \begin{cases} 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\ 0 & \text{otherwise} \end{cases}$$



$c_{ij} = 3$



$c_{ji} = 2$

- number of crossings of a permutations π_2:

$$\text{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji}) x_{ij} + \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji}$$

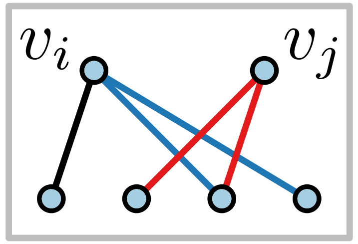
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

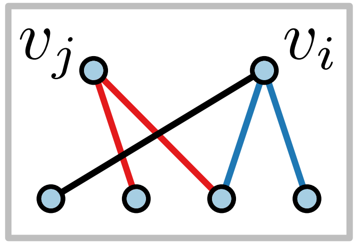
- constant $c_{ij} := \#$ crossings between edges incident to v_i and v_j when $\pi_2(v_i) < \pi_2(v_j)$

- variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$

$$x_{ij} = \begin{cases} 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\ 0 & \text{otherwise} \end{cases}$$



$c_{ij} = 3$



$c_{ji} = 2$

- number of crossings of a permutations π_2:

$$\text{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji}) x_{ij} + \underbrace{\sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji}}_{\text{constant}}$$

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

i.e., if $x_{ij} = 1$ and $x_{jk} = 1$, then $x_{ik} = 1$

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

i.e., if $x_{ij} = 1$ and $x_{jk} = 1$, then $x_{ik} = 1$

0 0 0

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

i.e., if $x_{ij} = \overset{1}{\underset{0}{}}$ and $x_{jk} = 1$, then $x_{ik} = 1$

i.e., if $x_{ij} = 0$ and $x_{jk} = 0$, then $x_{ik} = 0$

Properties.

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

i.e., if $x_{ij} = \overset{1}{\underset{0}{}}$ and $x_{jk} = \overset{1}{\underset{0}{}}$, then $x_{ik} = \overset{1}{\underset{0}{}}$

Properties.

- branch-and-cut technique applicable for this ILP

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

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

Properties.

- branch-and-cut technique applicable for this ILP
- useful for graphs of small to medium size

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

$$0 \le x_{ij} + x_{jk} - x_{ik} \le 1 \qquad \text{for } 1 \le i < j < k \le n_2$$

i.e., if $x_{ij} = \overset{}{\underset{0}{1}}$ and $x_{jk} = \underset{0}{1}$, then $x_{ik} = \underset{0}{1}$

Properties.

- branch-and-cut technique applicable for this ILP
- useful for graphs of small to medium size
- finds optimal solution

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

- objective (minimize the number of crossings):

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

- transitivity constraints:

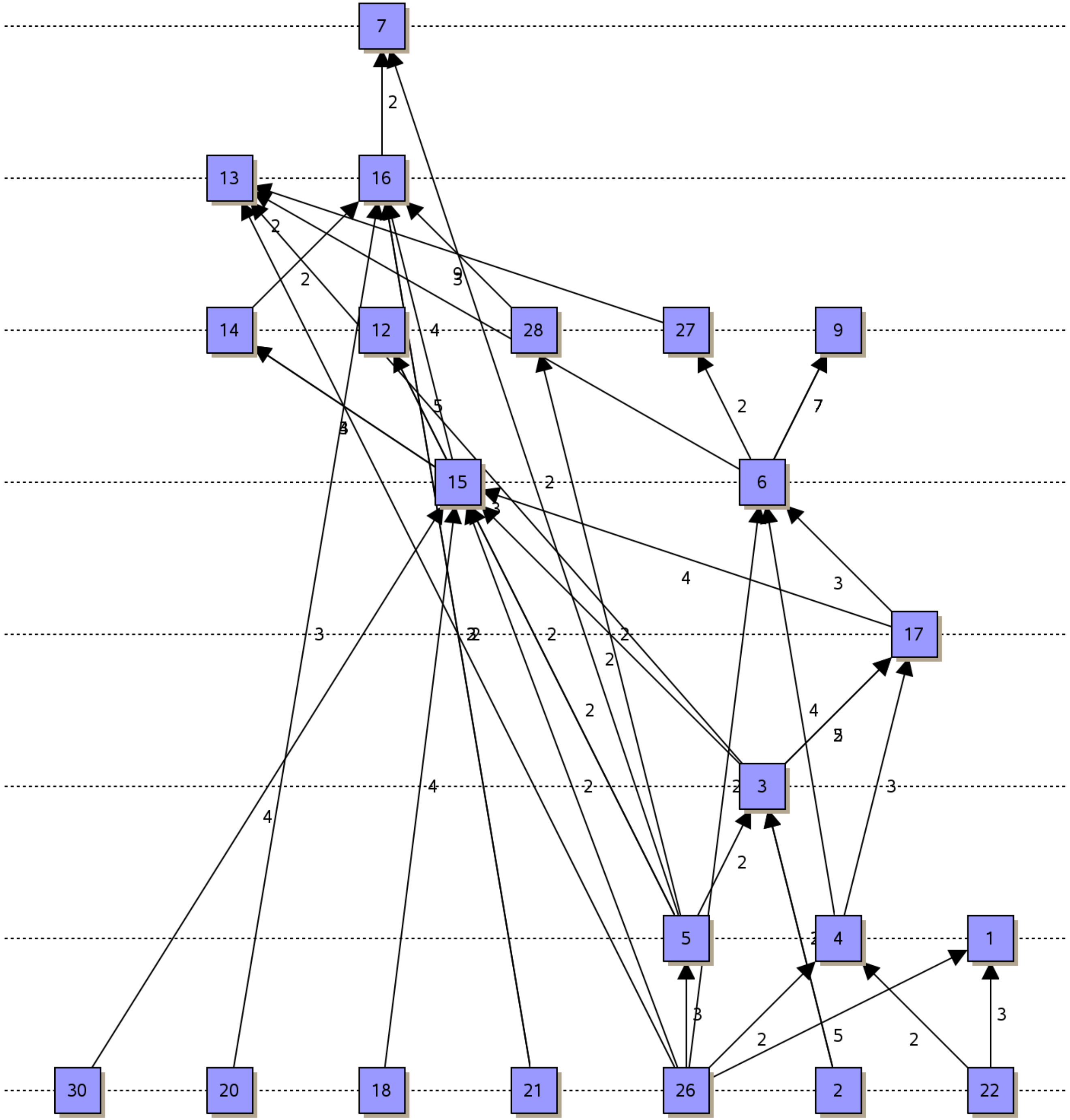
$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

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

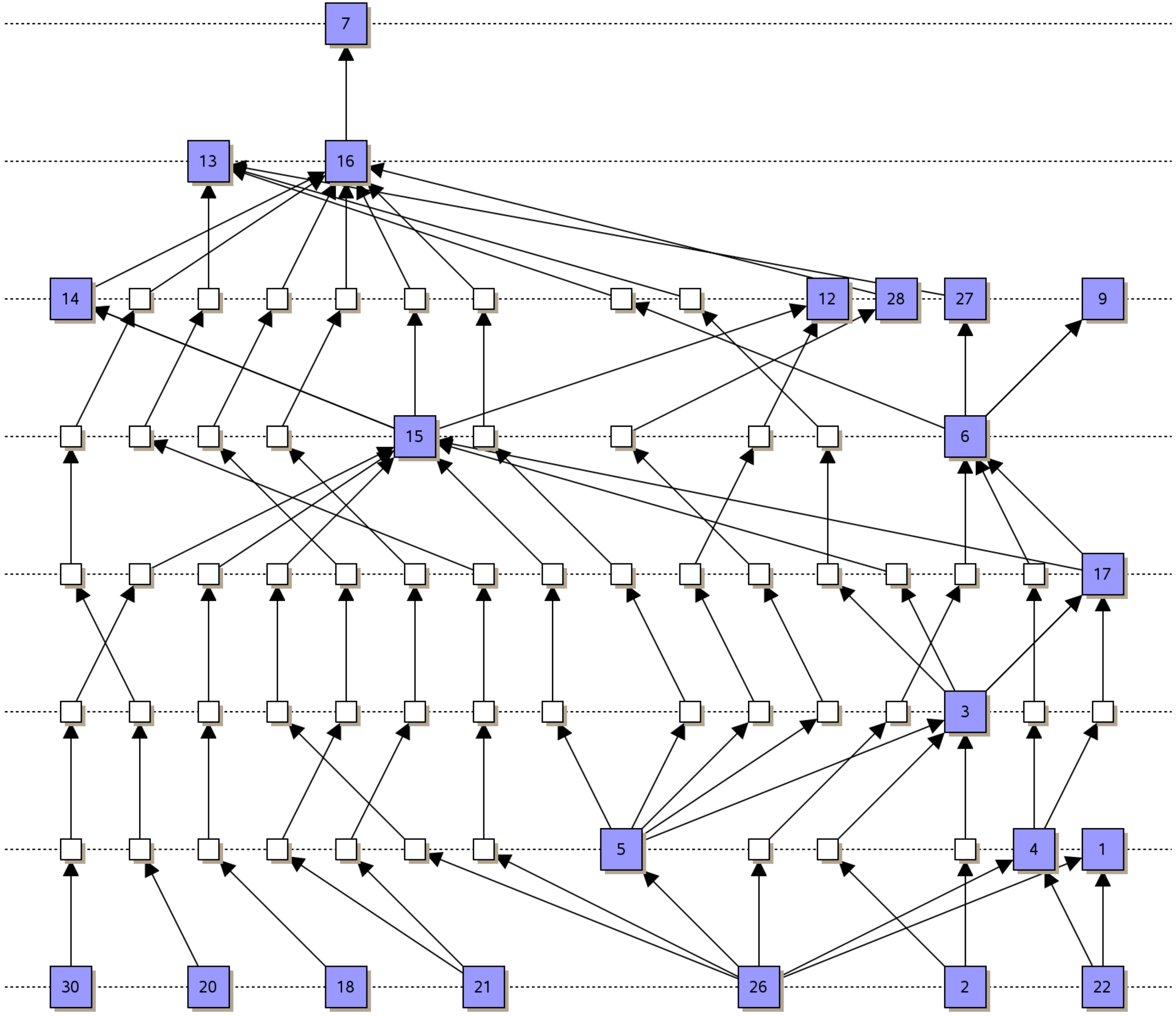
Properties.

- branch-and-cut technique applicable for this ILP
- useful for graphs of small to medium size
- finds optimal solution
- solution in polynomial time is not guaranteed

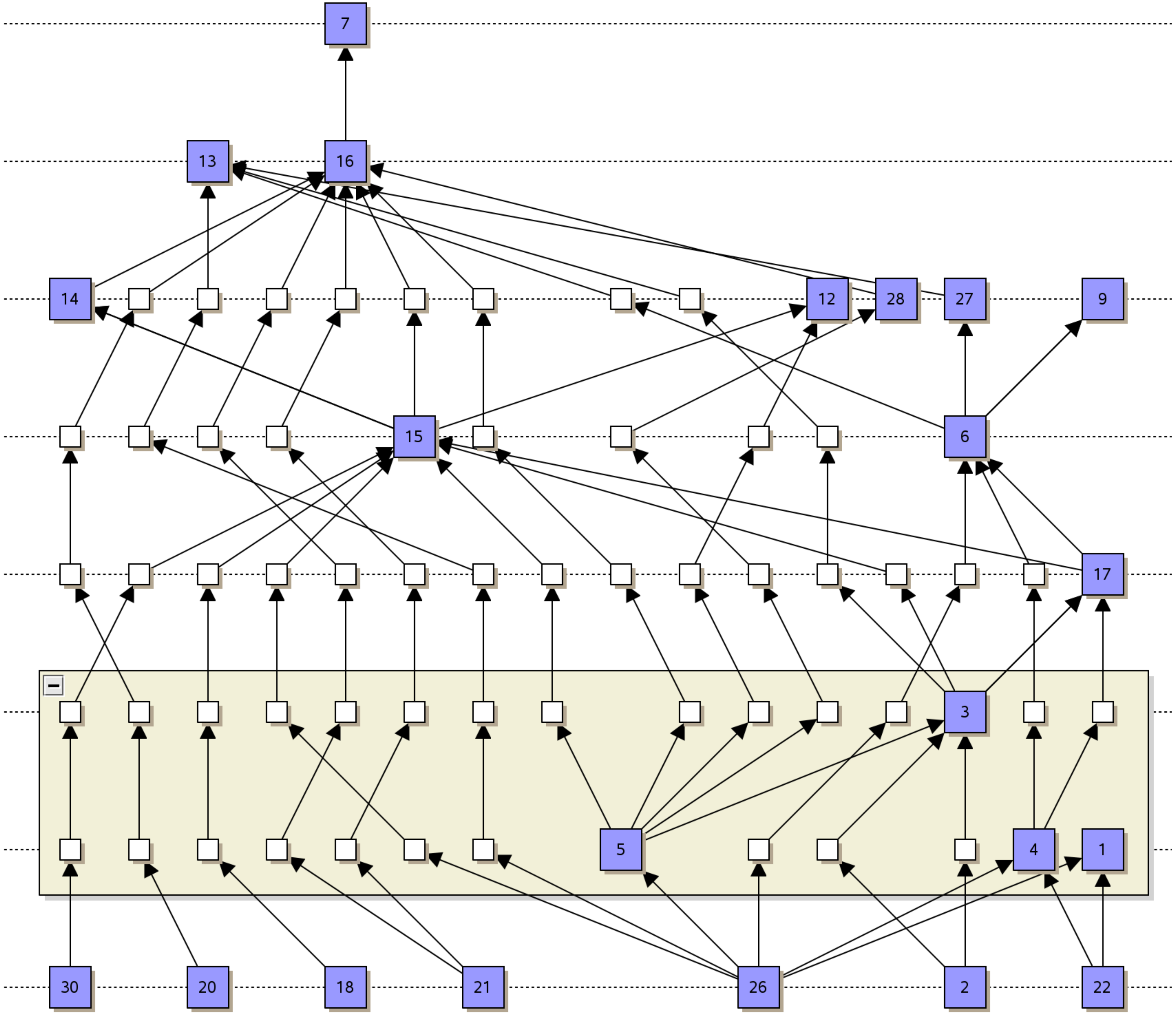
Iterations on Example



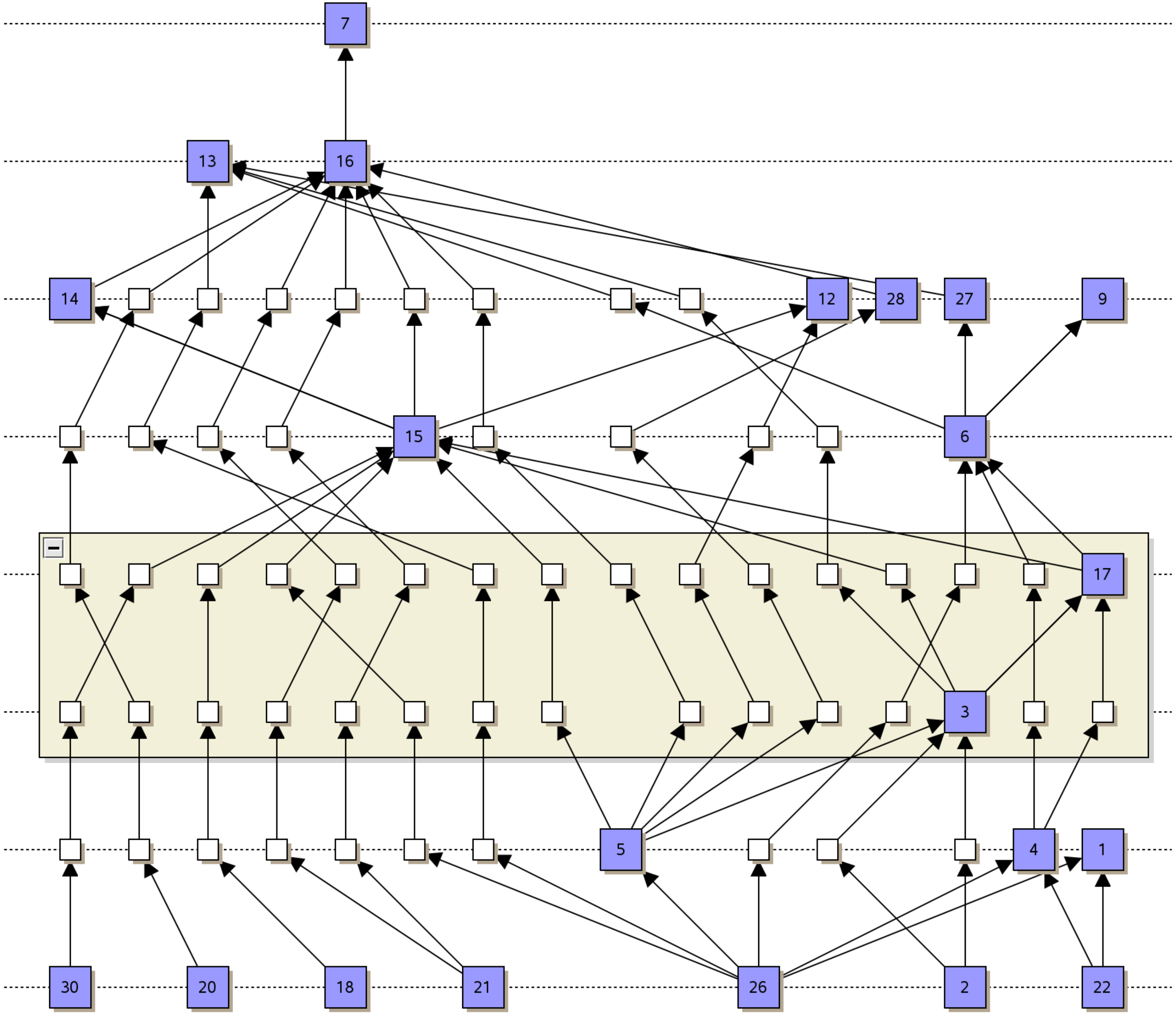
Iterations on Example



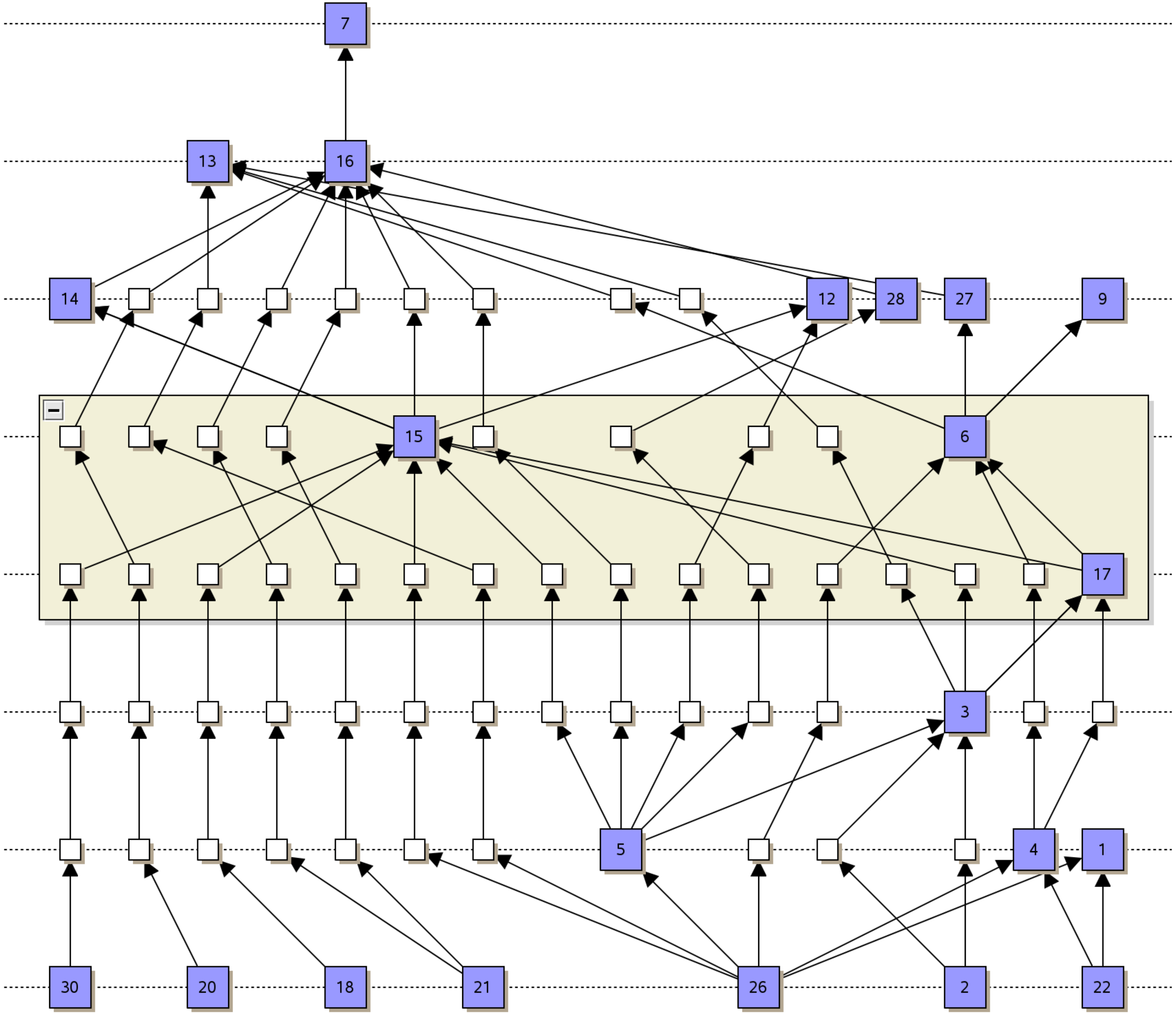
Iterations on Example



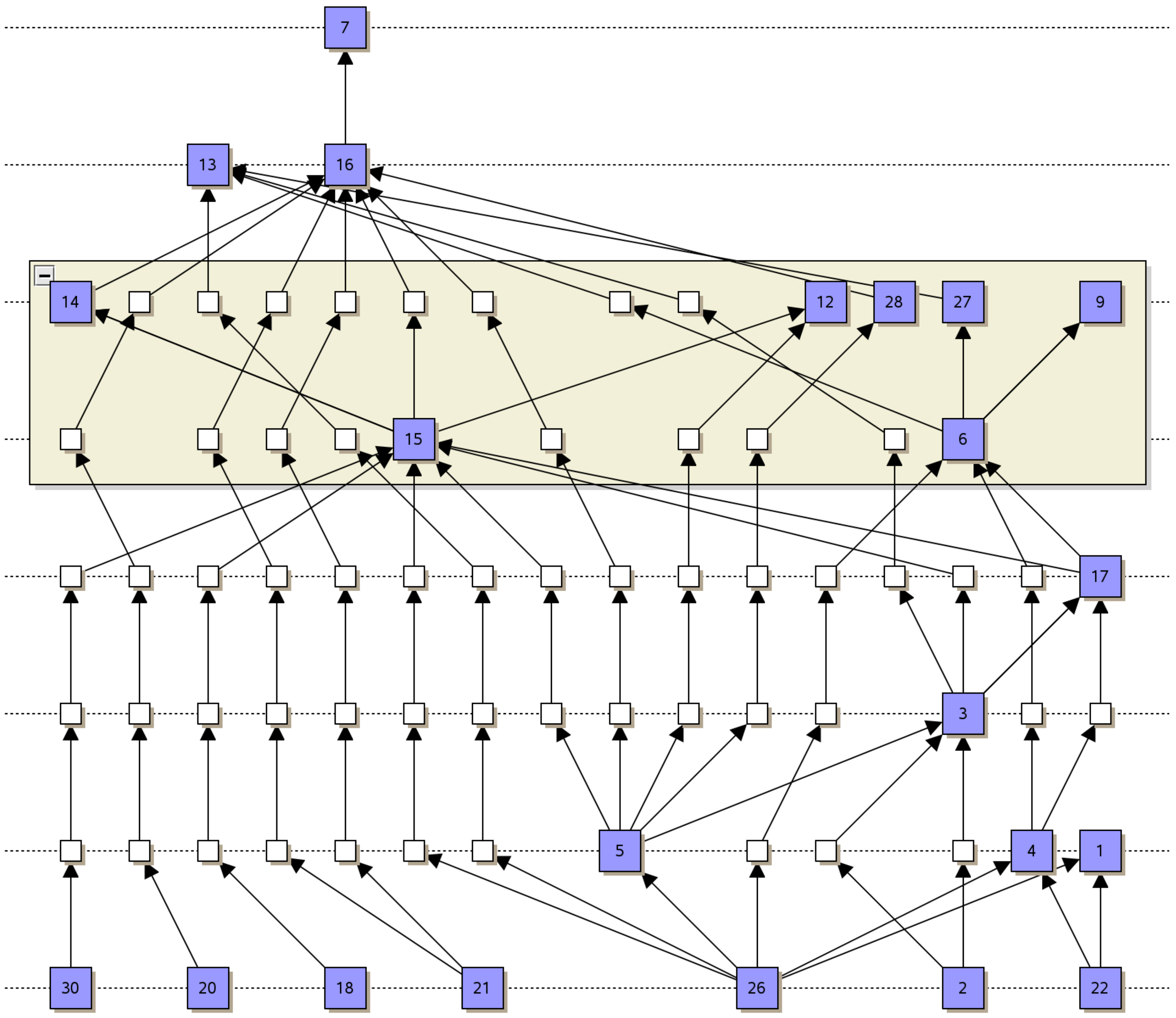
Iterations on Example



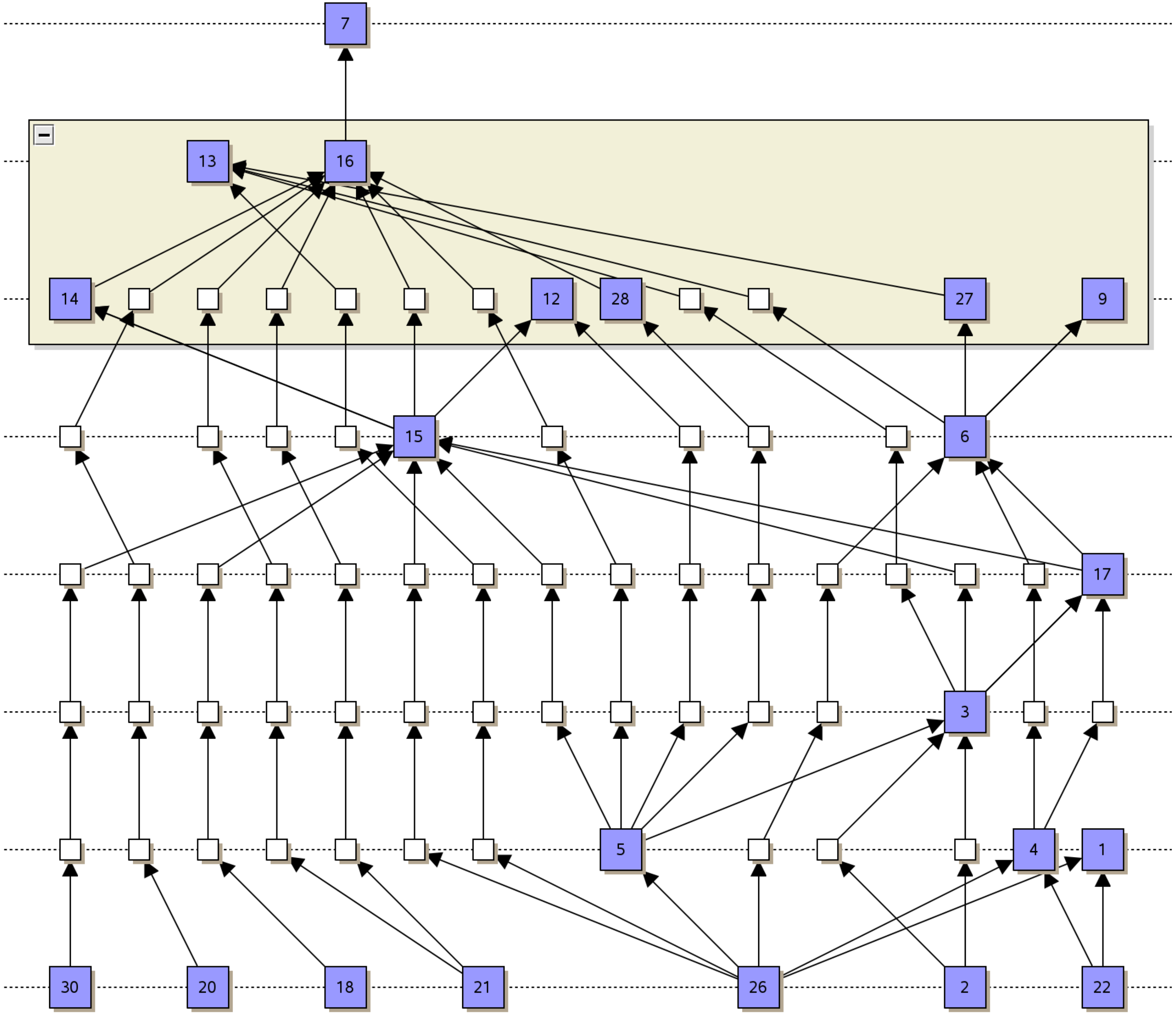
Iterations on Example



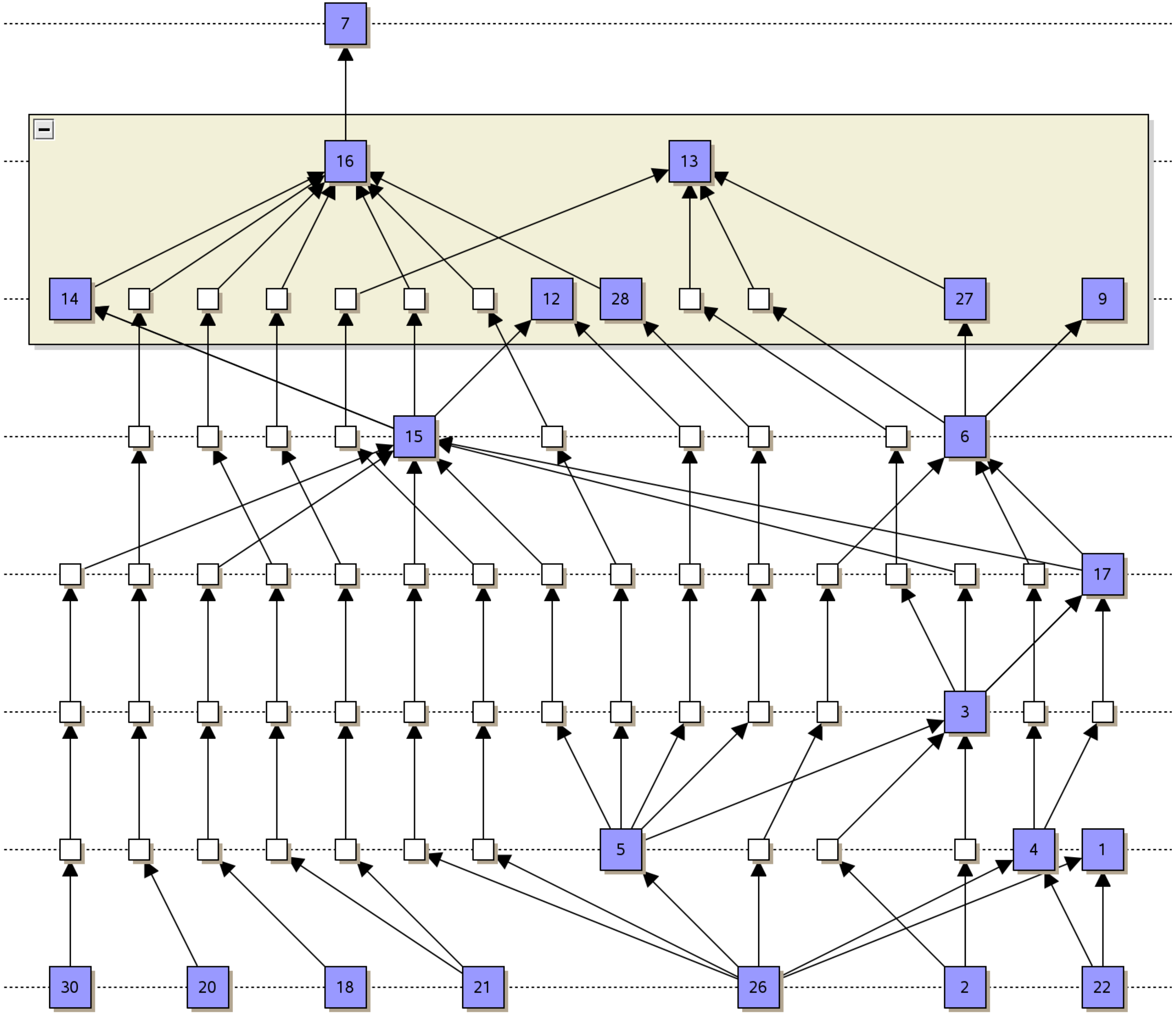
Iterations on Example



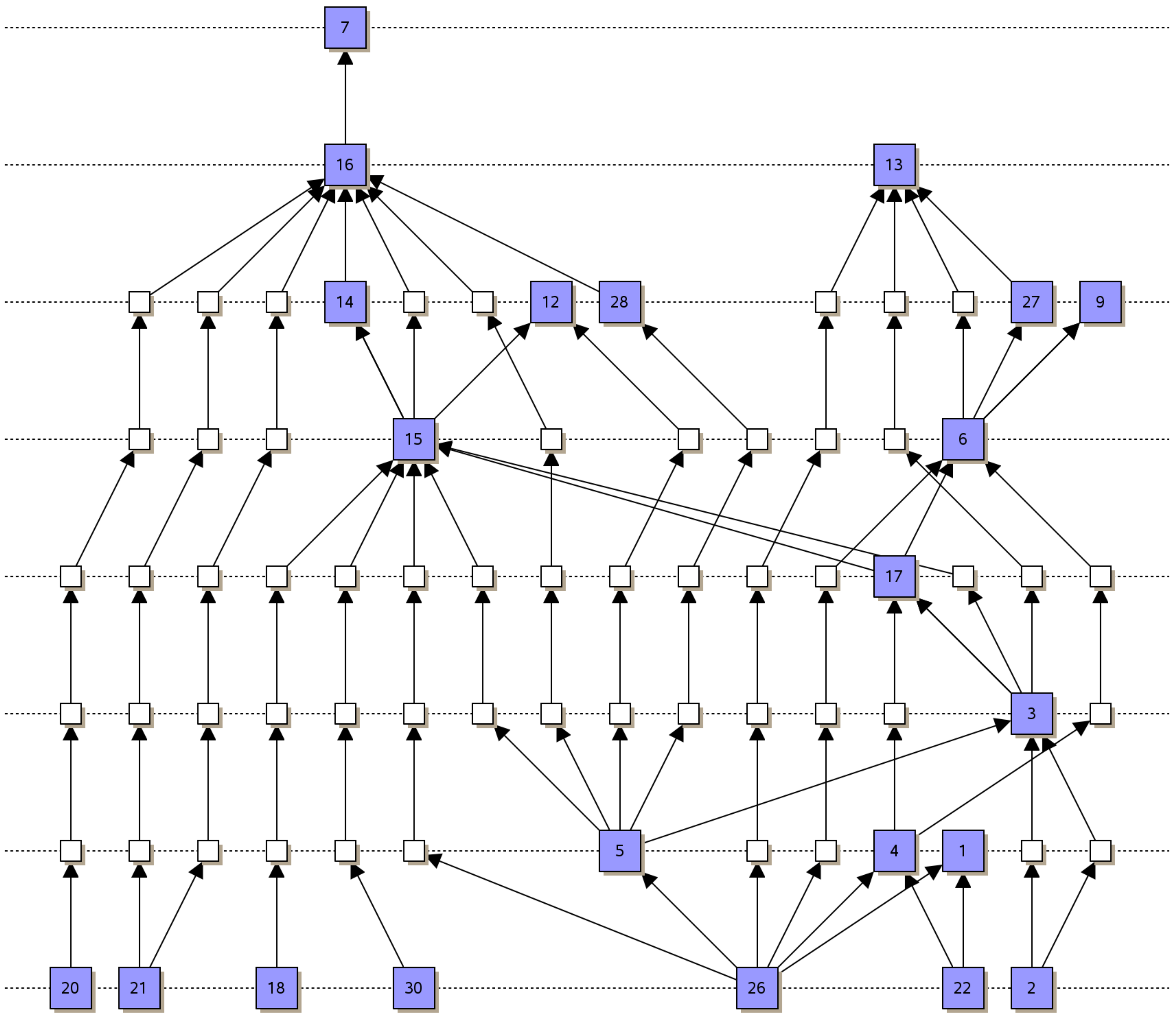
Iterations on Example



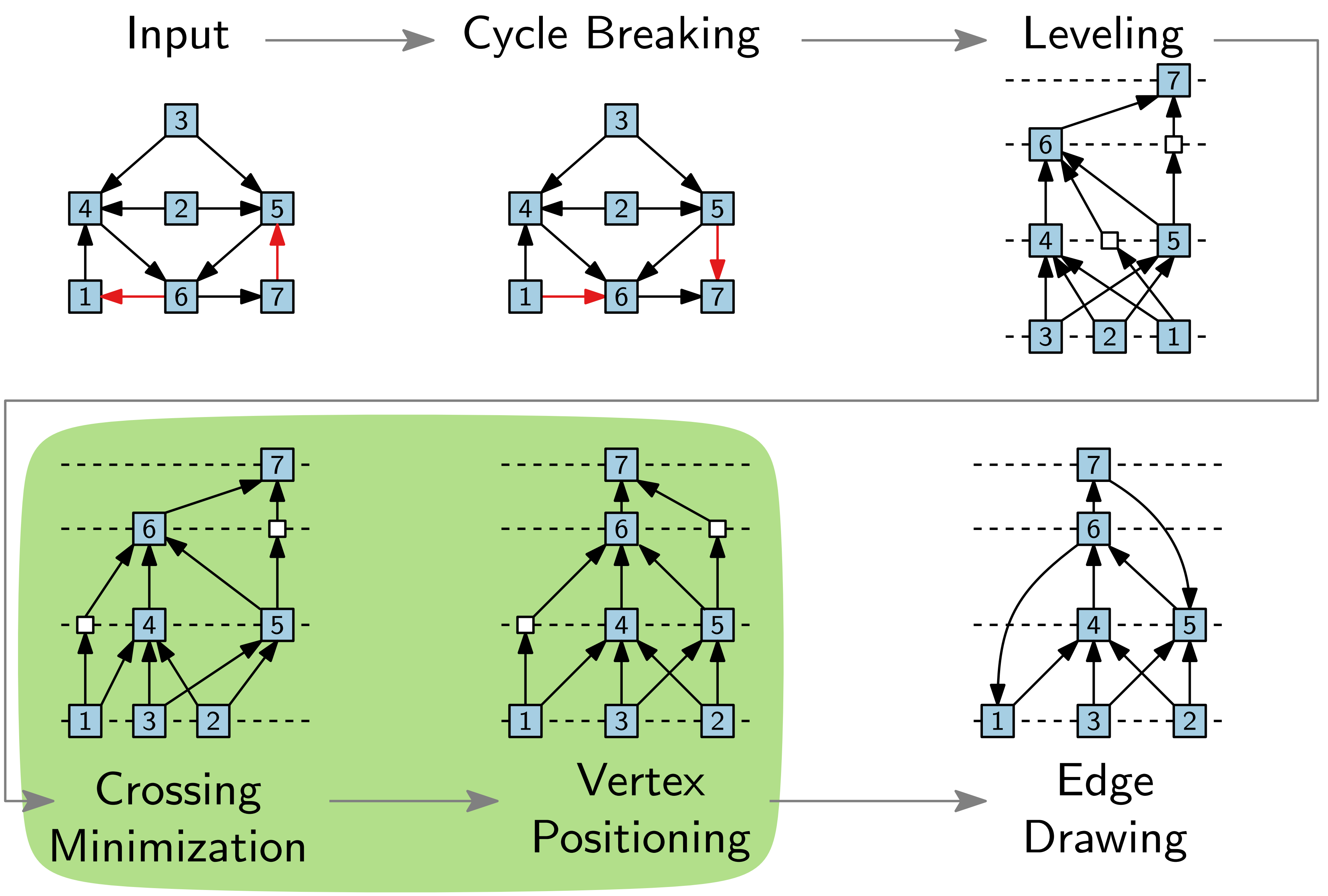
Iterations on Example



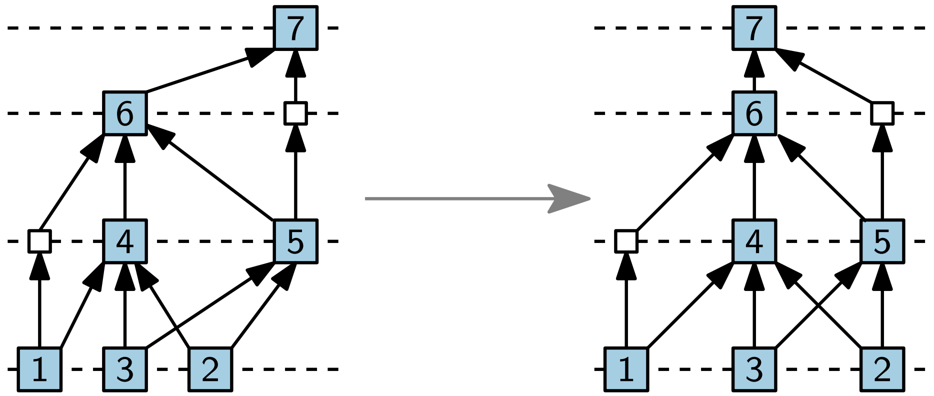
Iterations on Example



Step 4: Vertex Positioning



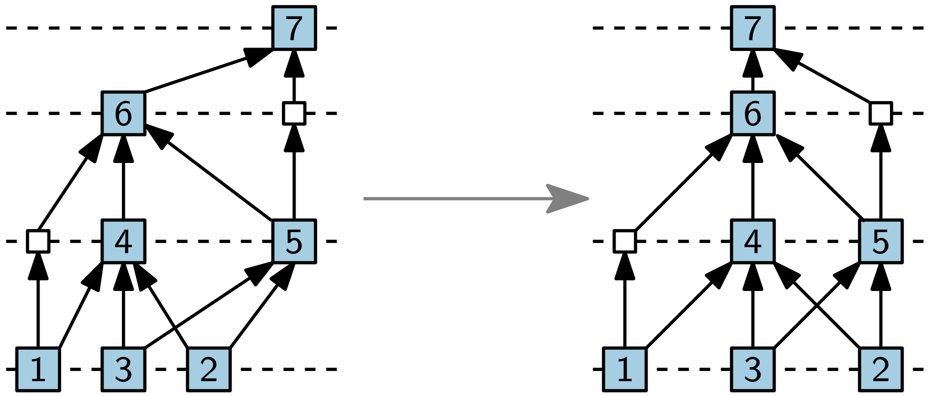
Step 4: Vertex Positioning



Step 4: Vertex Positioning

Goals.

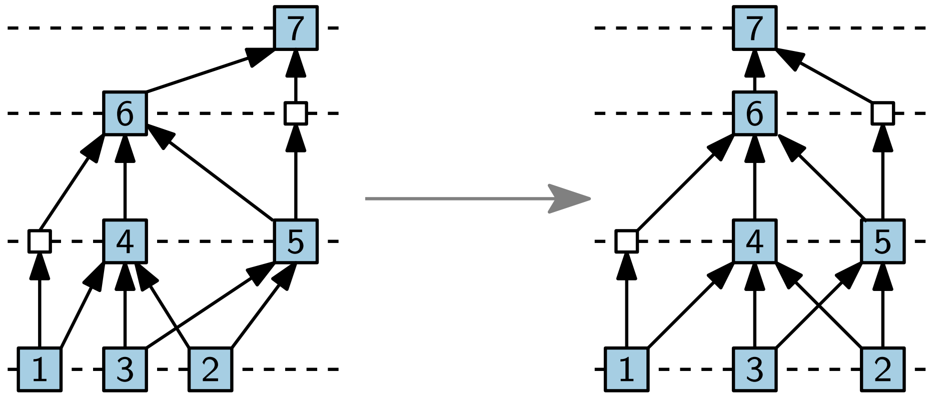
- paths of a single edge should be (close to) straight
- vertices on a layer evenly spaced
- perfer vertical edges



Step 4: Vertex Positioning

Goals.

- paths of a single edge should be (close to) straight
- vertices on a layer evenly spaced
- perfer vertical edges

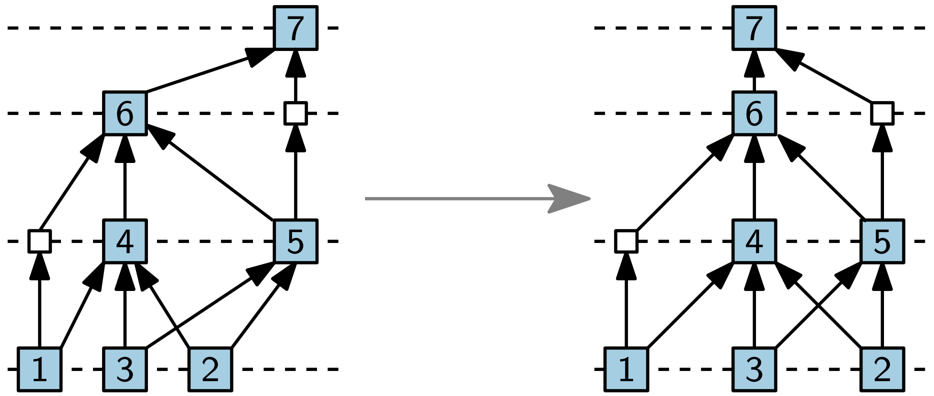


- **Exact:** Quadratic Program (QP)

Step 4: Vertex Positioning

Goals.

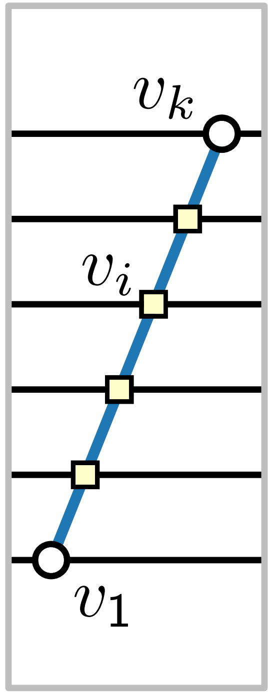
- paths of a single edge should be (close to) straight
- vertices on a layer evenly spaced
- perfer vertical edges



- **Exact:** Quadratic Program (QP)
- **Heuristic:** Iterative approach

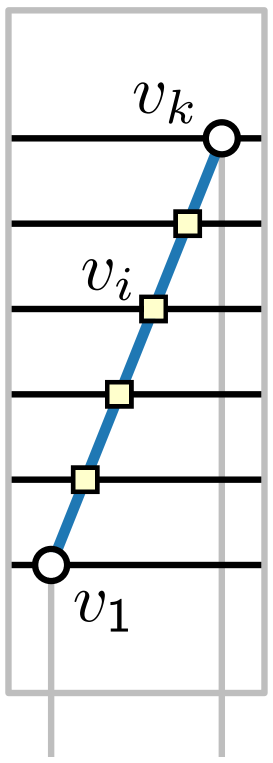
Quadratic Program

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



Quadratic Program

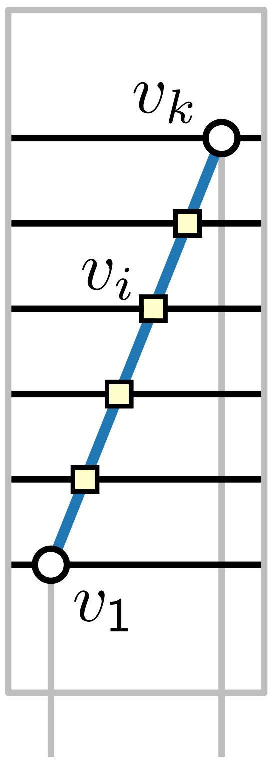
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):



Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

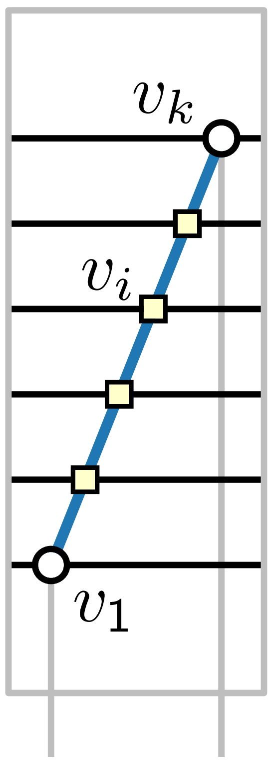
$$\overline{x(v_i)} =$$



Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

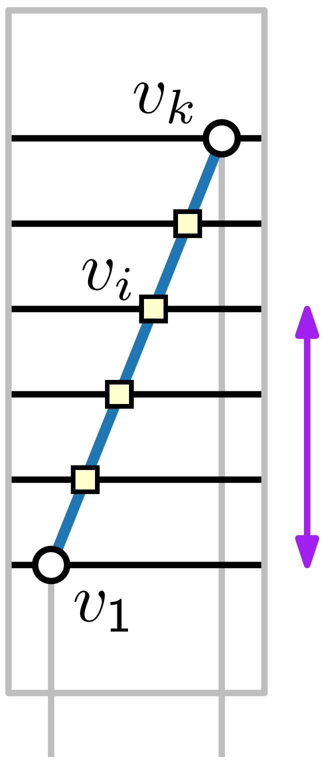
$$\overline{x(v_i)} = x(v_1) +$$



Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

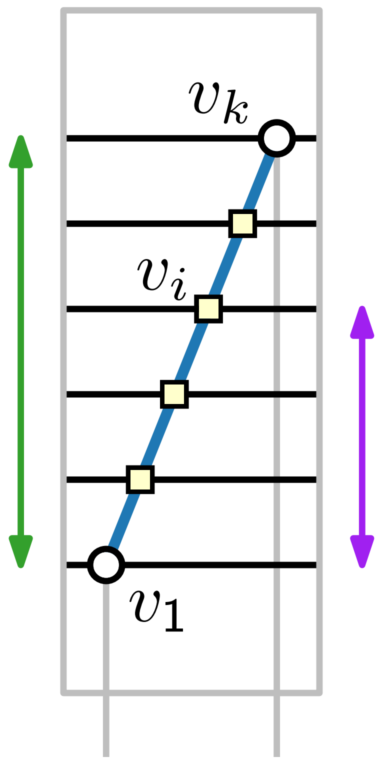
$$\overline{x(v_i)} = x(v_1) + \frac{i-1}{}$$



Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

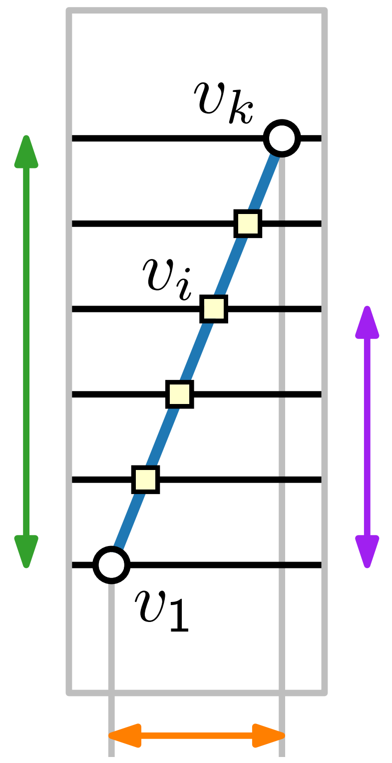
$$\overline{x(v_i)} = x(v_1) + \frac{i-1}{k-1}$$



Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

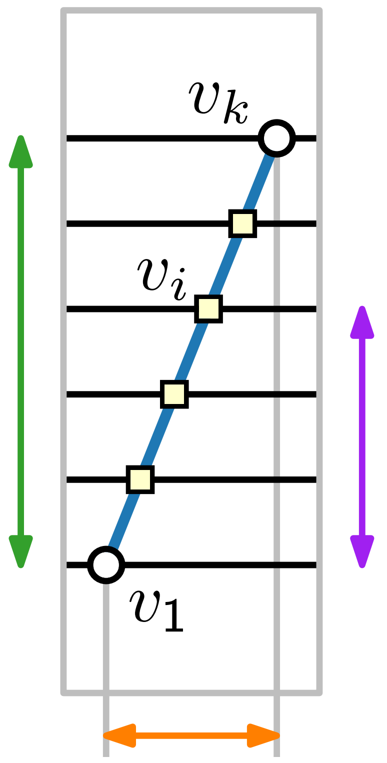


Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line



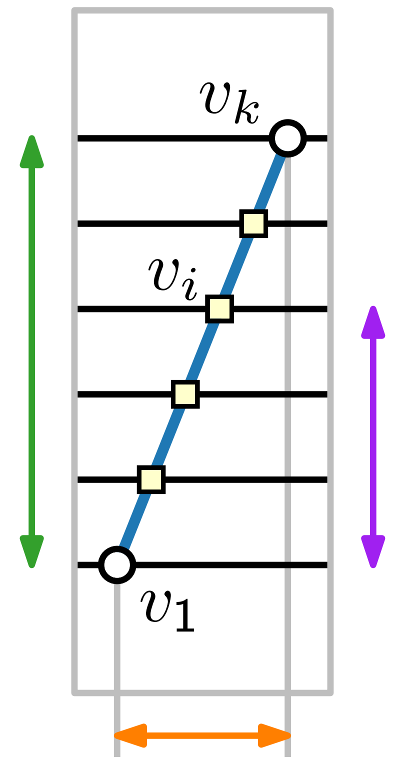
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

$$\mathrm{dev}(p_e) :=$$



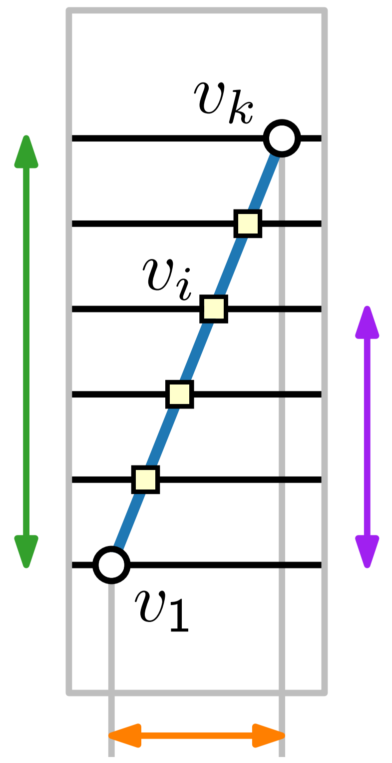
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

$$\mathrm{dev}(p_e) := \sum_{i=2}^{k-1}$$



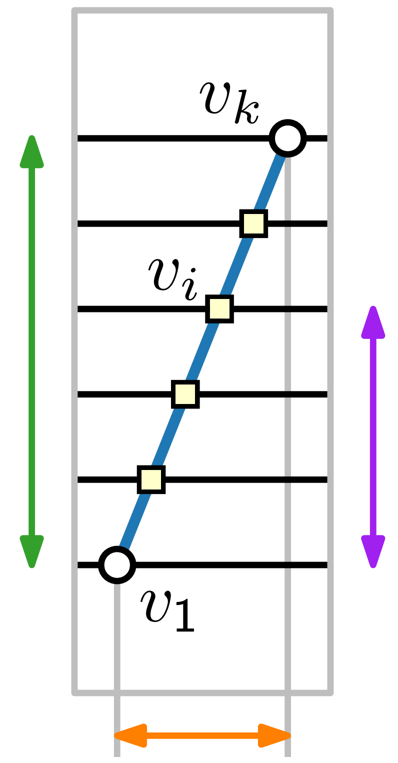
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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



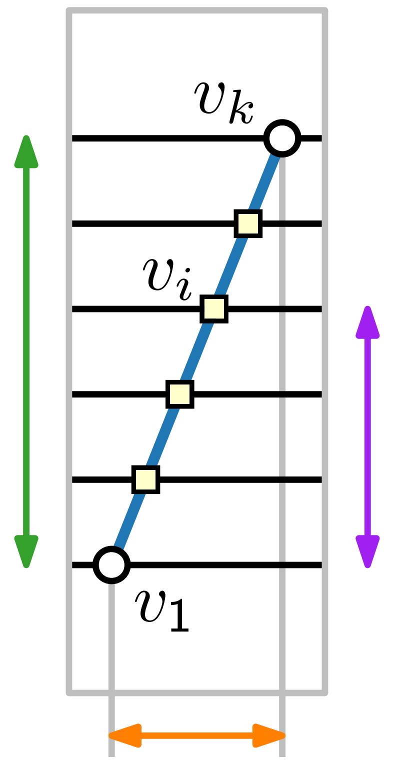
Quadratic Program

- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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



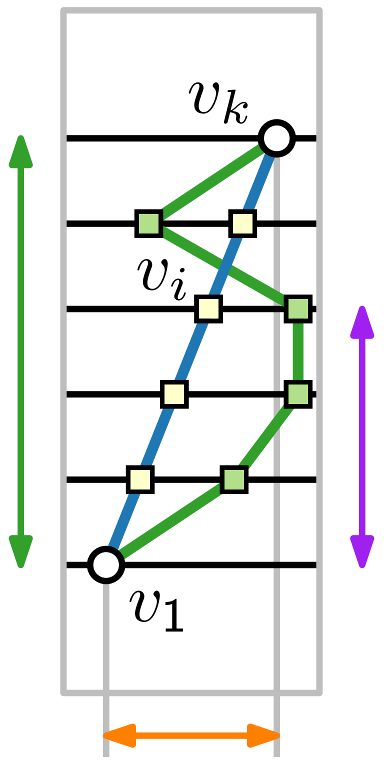
Quadratic Program

- Consider the path $p_e = (v_1, \dots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \dots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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



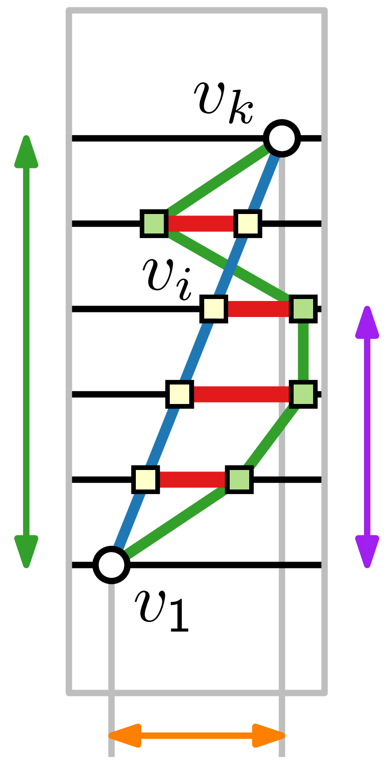
Quadratic Program

- Consider the path $p_e = (v_1, \dots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \dots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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



Quadratic Program

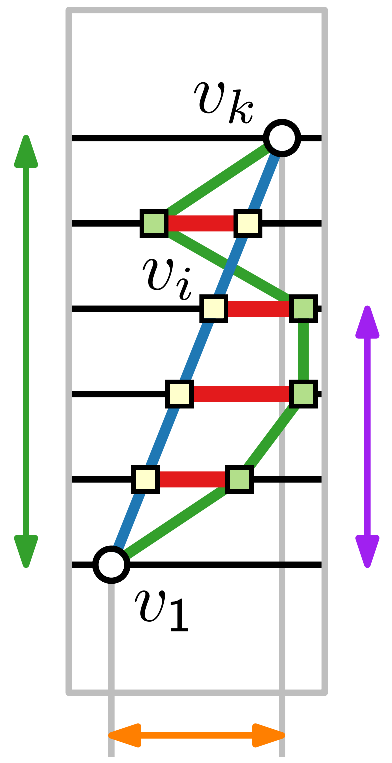
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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

- Objective function:



Quadratic Program

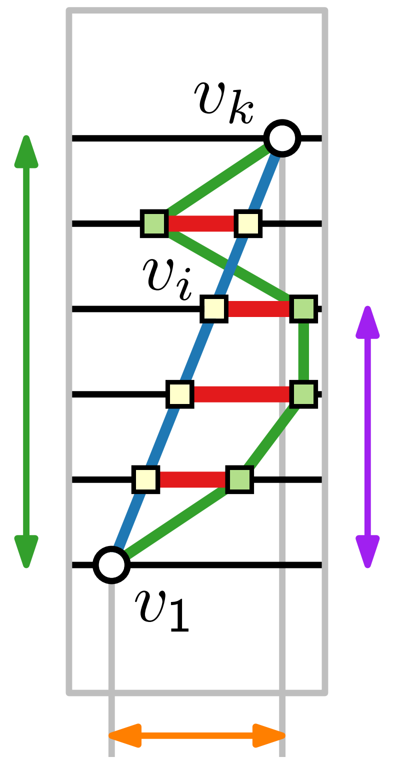
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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

- Objective function: $\min \sum_{e \in E} \mathrm{dev}(p_e)$



Quadratic Program

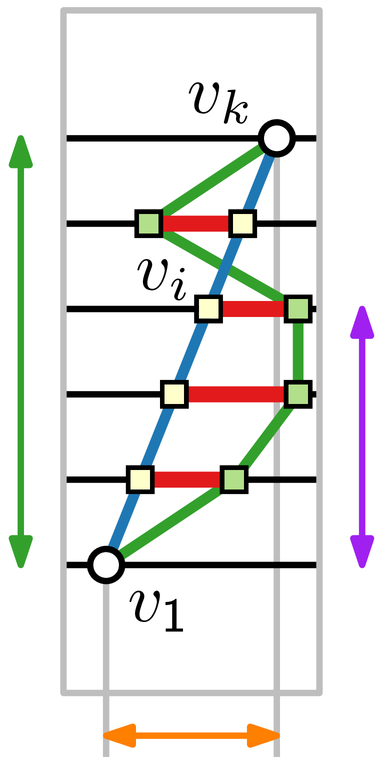
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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

- Objective function: $\min \sum_{e \in E} \mathrm{dev}(p_e)$
- Constraints for all vertices v, w in the same layer with w to the right of v:



Quadratic Program

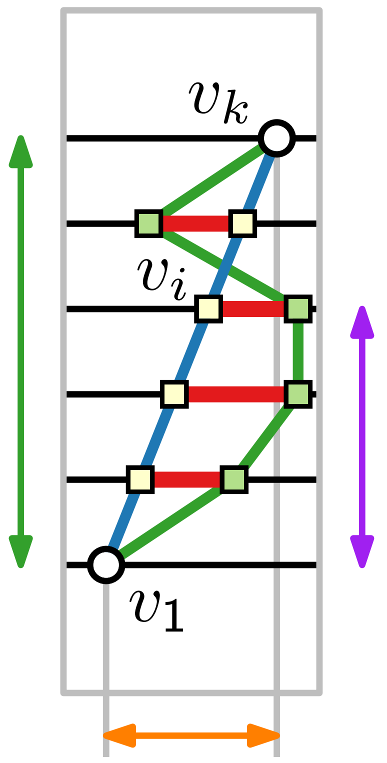
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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

- Objective function: $\min \sum_{e \in E} \mathrm{dev}(p_e)$
- Constraints for all vertices v, w in the same layer with w to the right of v: $x(w) - x(v) \geq \rho$



Quadratic Program

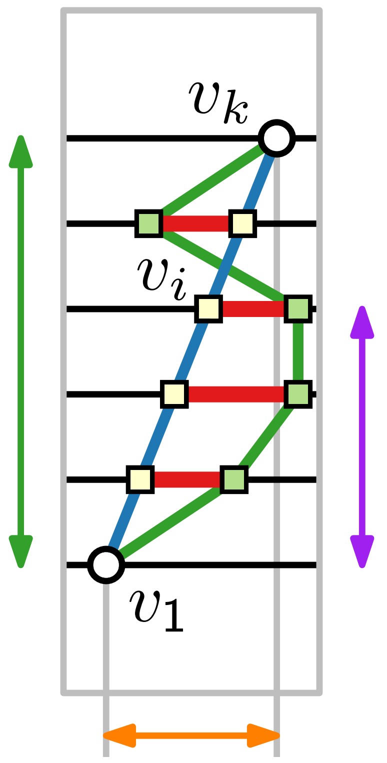
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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

- Objective function: $\min \sum_{e \in E} \mathrm{dev}(p_e)$
- Constraints for all vertices v, w in the same layer with w to the right of v: $x(w) - x(v) \geq \rho$ ← min. horizontal distance



Quadratic Program

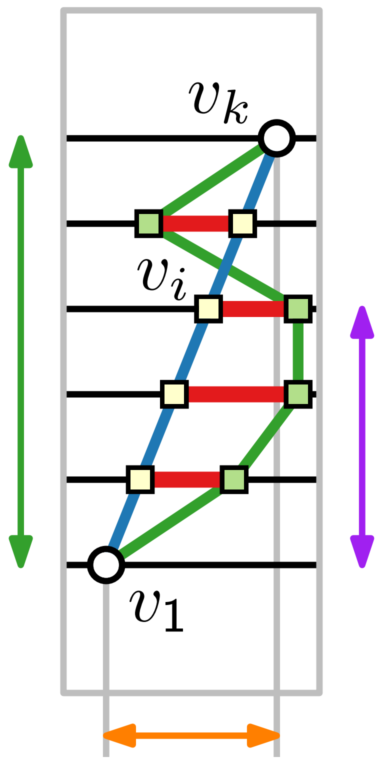
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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

- Objective function: $\min \sum_{e \in E} \operatorname{dev}(p_e)$
- Constraints for all vertices v, w in the same layer with w to the right of v: $x(w) - x(v) \geq \rho$ ← min. horizontal distance



- QP is time-expensive

Quadratic Program

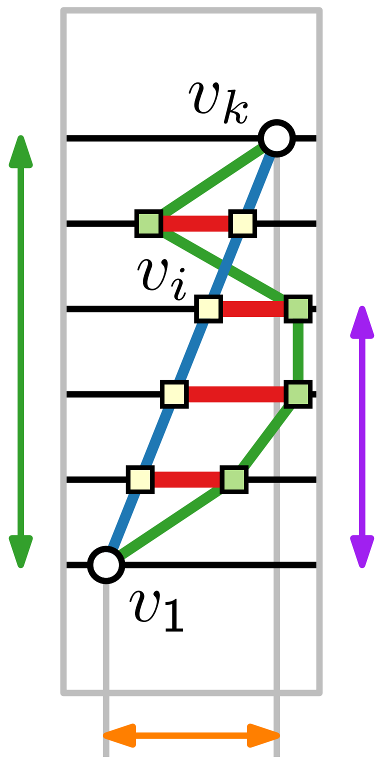
- Consider the path $p_e = (v_1, \ldots, v_k)$ of an edge $e = v_1 v_k$ with dummy vertices: $v_2, \ldots, v_{k-1}$
- x-coordinate of v_i according to the line $\overline{v_1 v_k}$ (with equal spacing):

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

- Define the deviation from the line

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

- Objective function: $\min \sum_{e \in E} \mathrm{dev}(p_e)$
- Constraints for all vertices v, w in the same layer with w to the right of v: $x(w) - x(v) \geq \rho$ ← min. horizontal distance



- QP is time-expensive
- width can be exponential

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

Iterative Heuristic

- Compute an initial layout
- Apply the following steps as long as improvements can be made:

  1. vertex positioning
  2. edge straightening

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

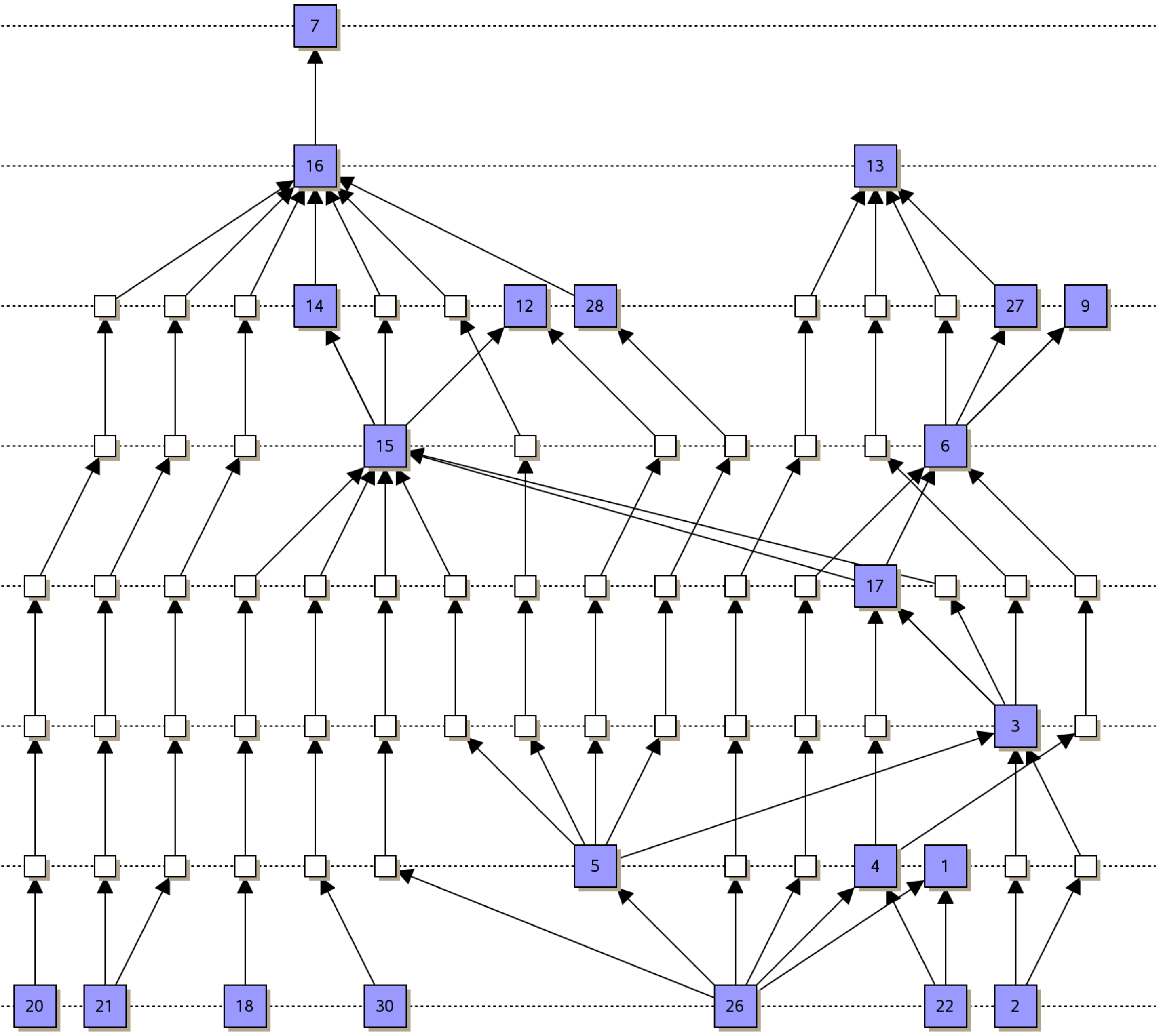
Iterative Heuristic

- Compute an initial layout
- Apply the following steps as long as improvements can be made:

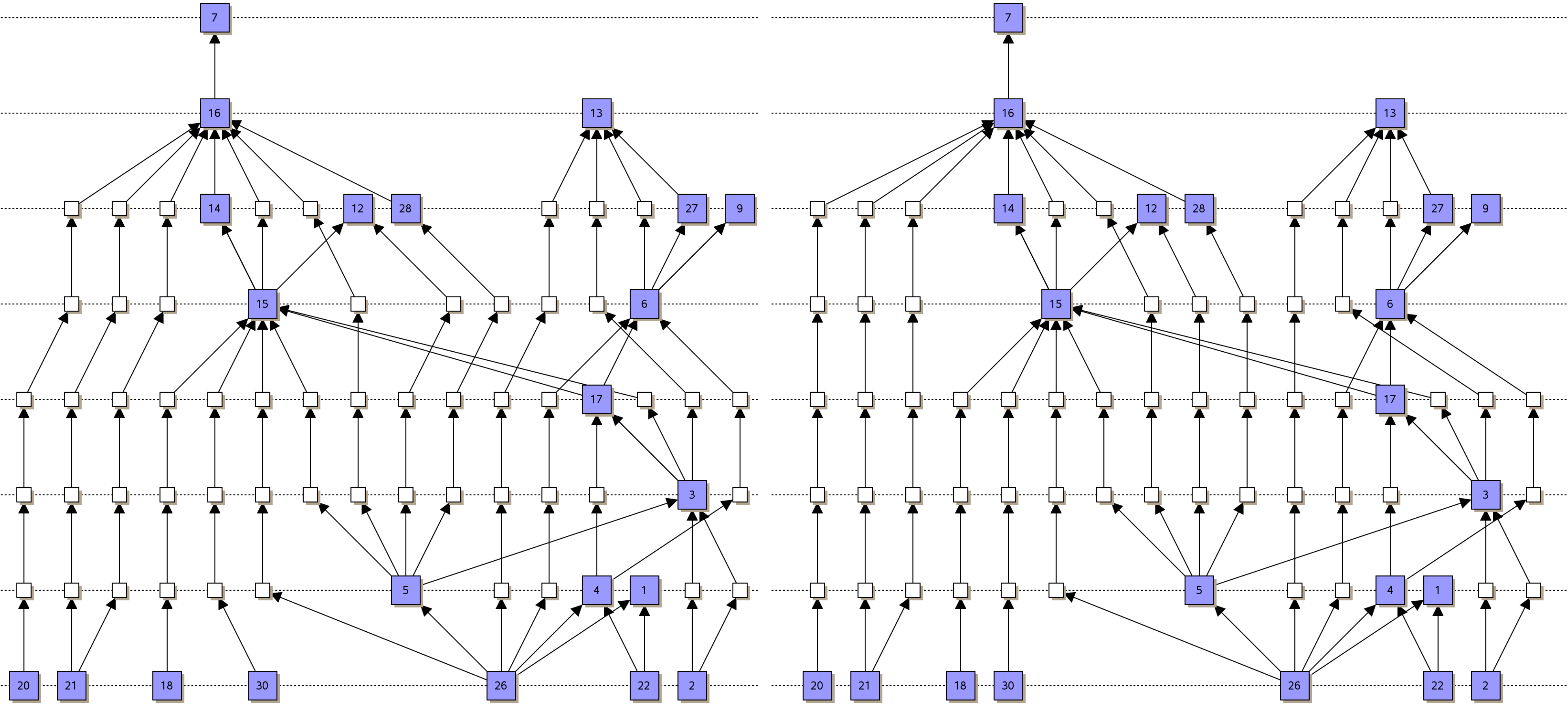
  1. vertex positioning
  2. edge straightening
  3. compactifying the layout width

- Other algorithms include the algorithm by Brandes and Köpf ’02:
  - tries to align vertices vertically
  - does horizontal compaction afterwards
  - linear running time

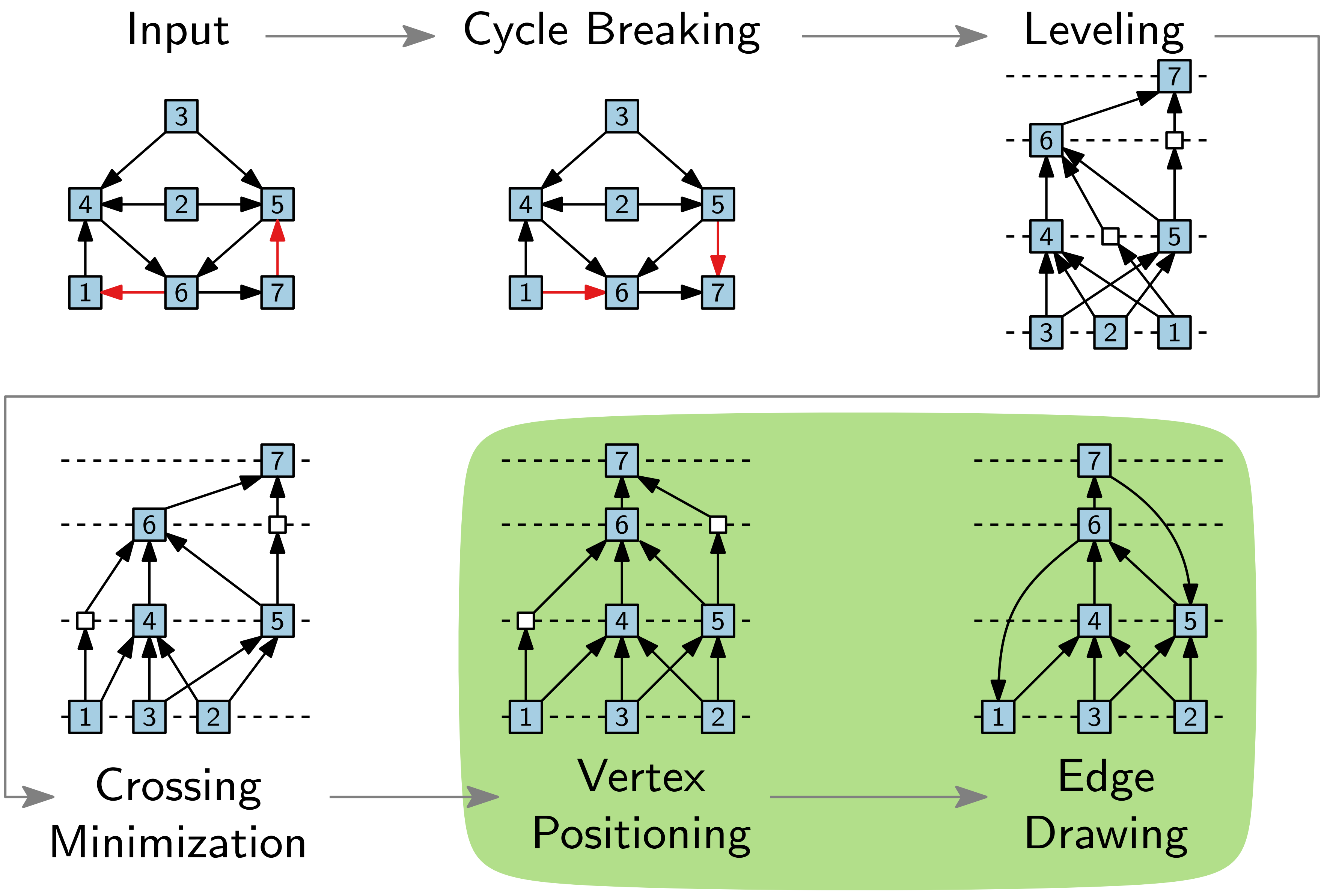
Example



Example

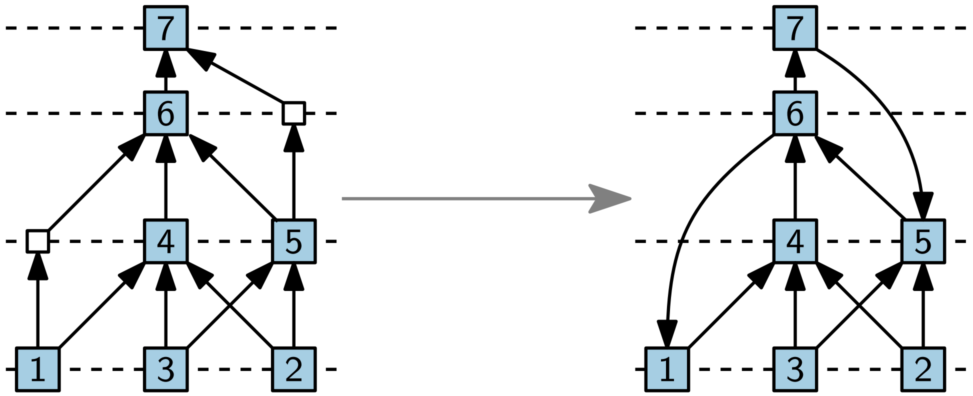

7
16
13
14
12
28
27
9
15
6
17
3
5
4
1
20
21
18
30
26
22
2
7
16
13
14
12
28
27
9
15
6
17
3
5
4
1
20
21
18
30
26
22
2

Step 5: Drawing Edges



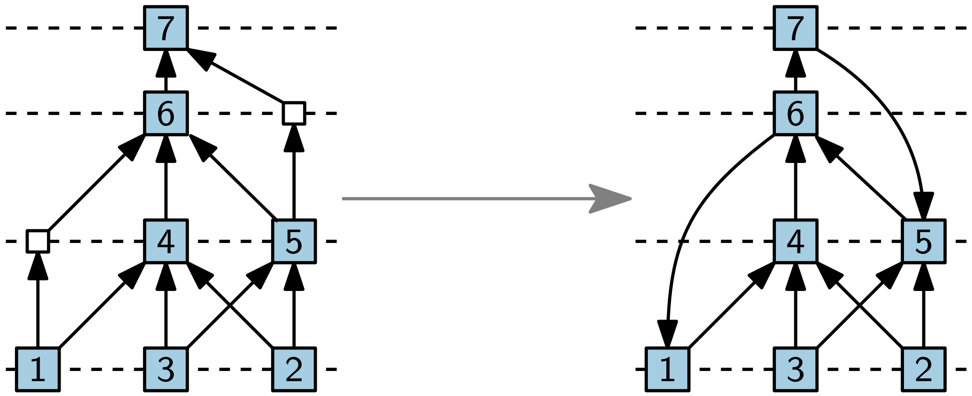
Step 5: Drawing Edges

Possibility.
Substitute polylines by Bézier curves.



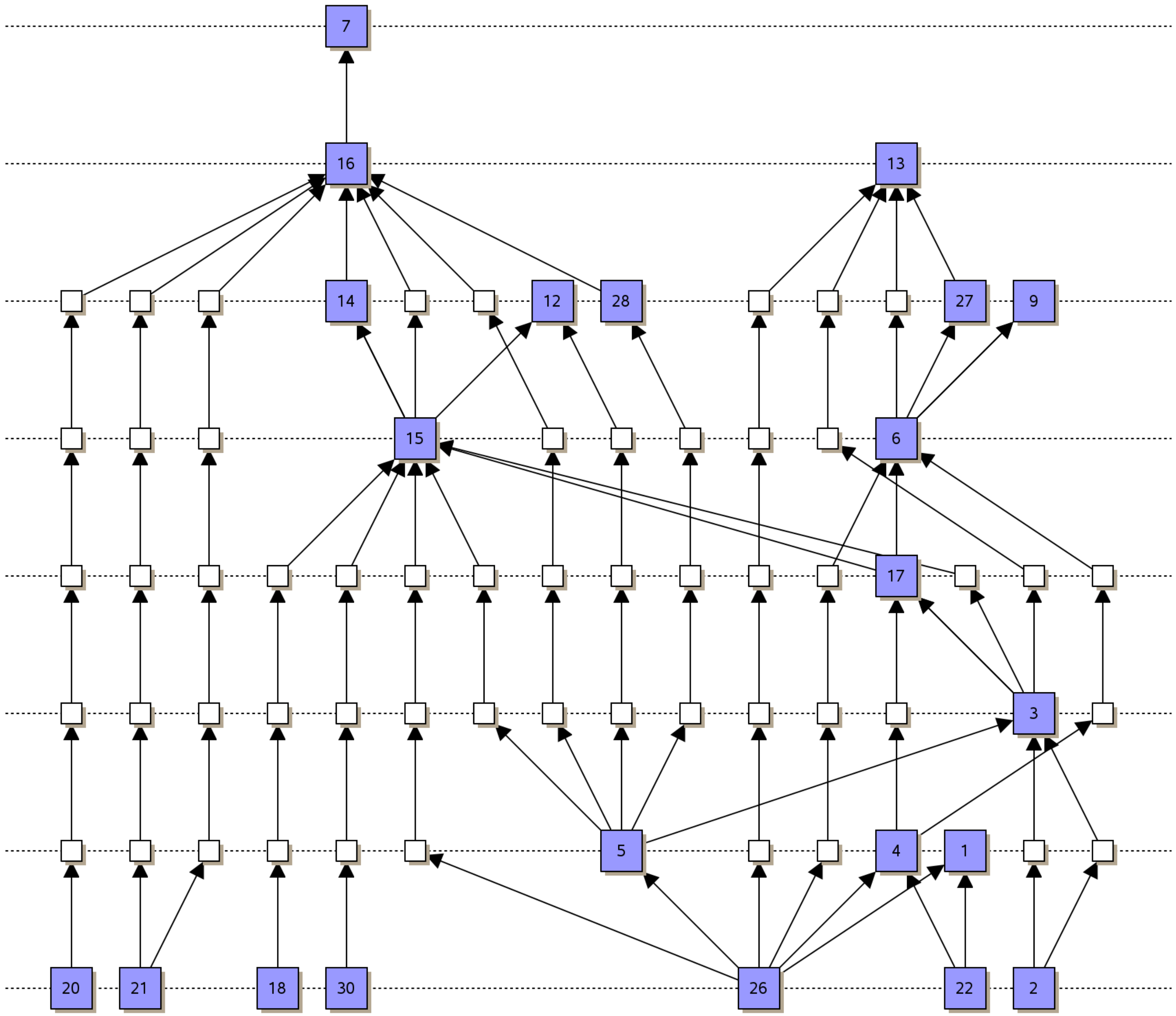
Step 5: Drawing Edges

Possibility.
Substitute polylines by Bézier curves.

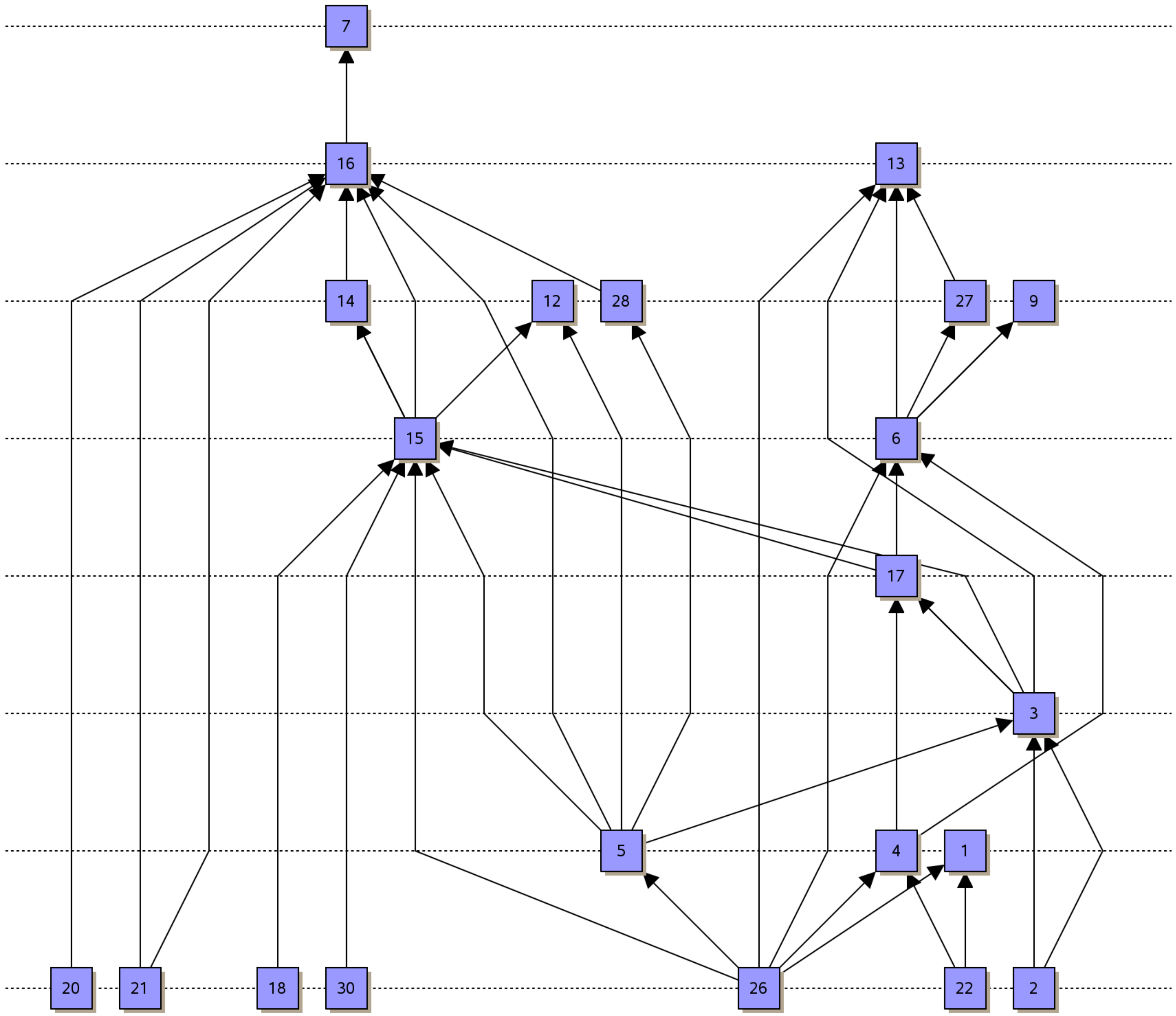


Remark.
Draw reversed edges downwards.

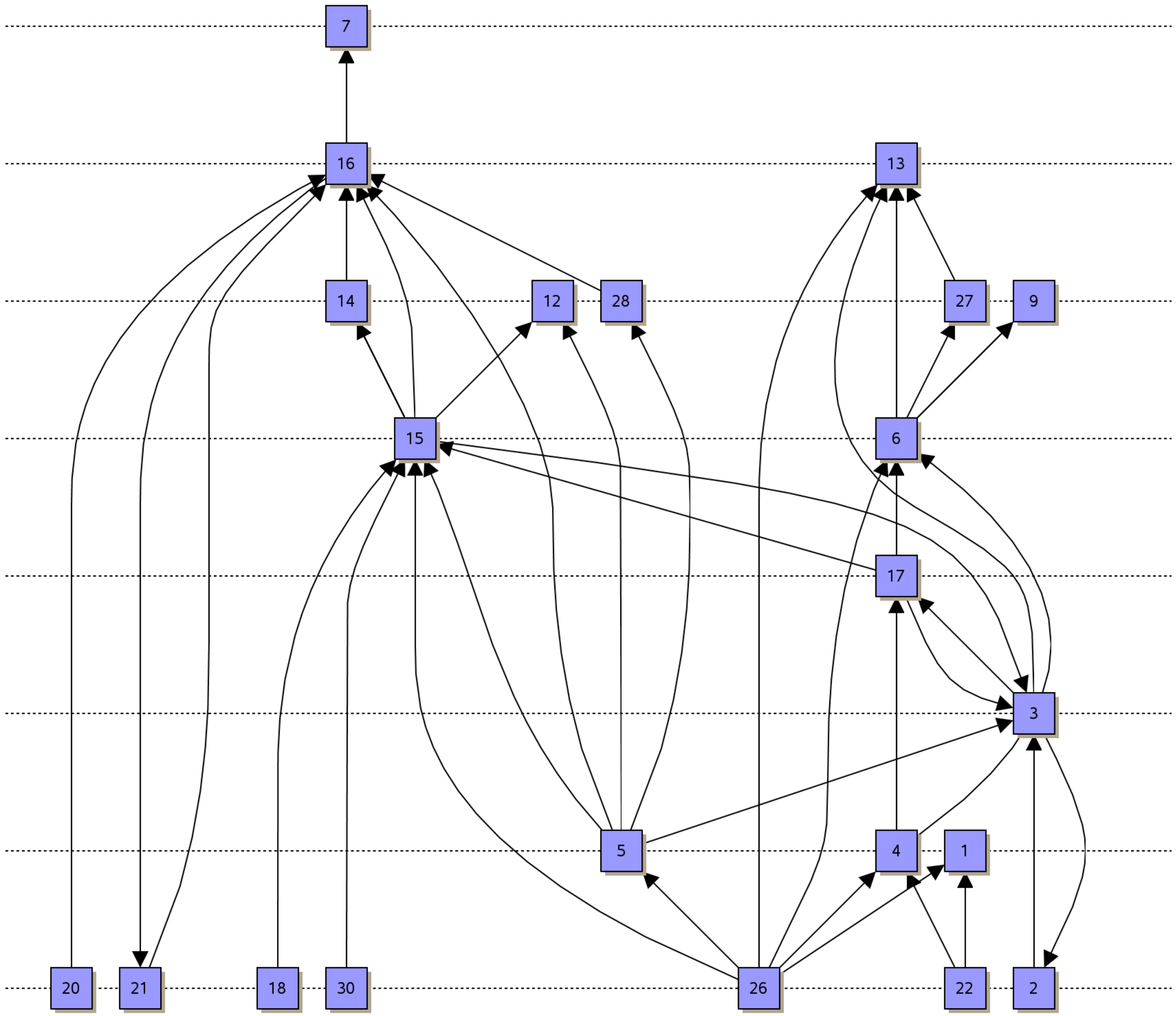
Example



Example

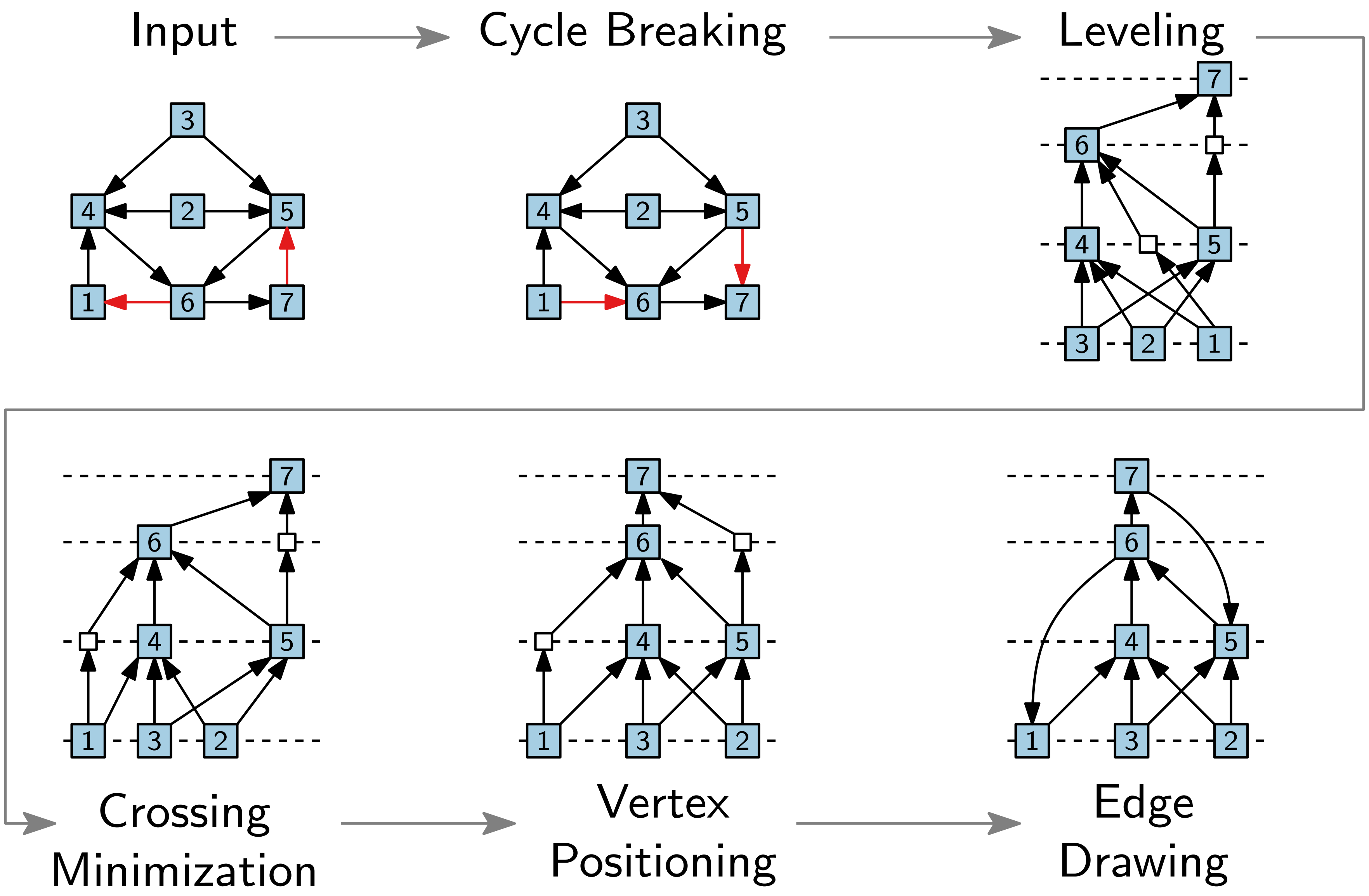


Example



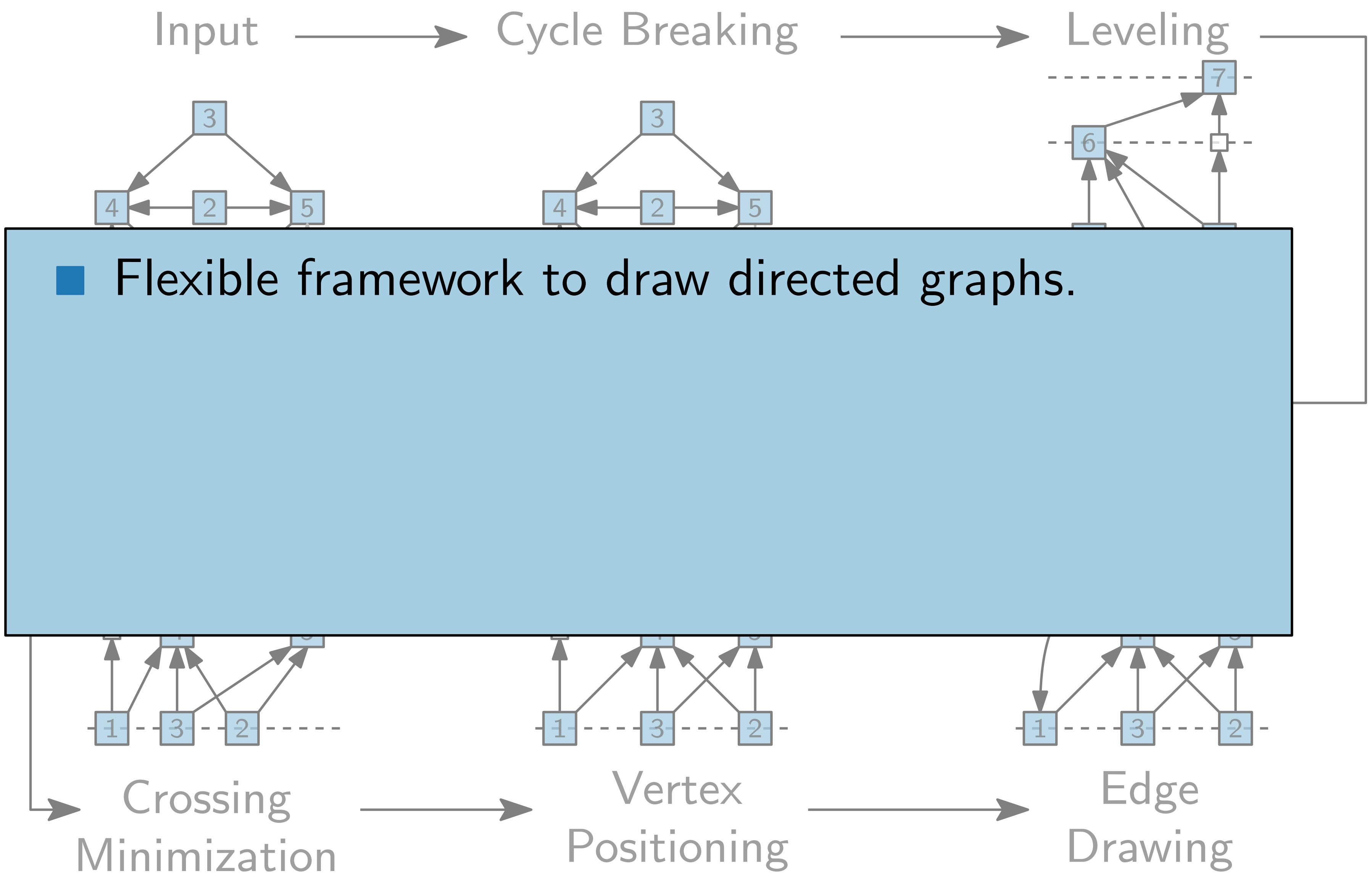
Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]



Classical Approach – Sugiyama Framework

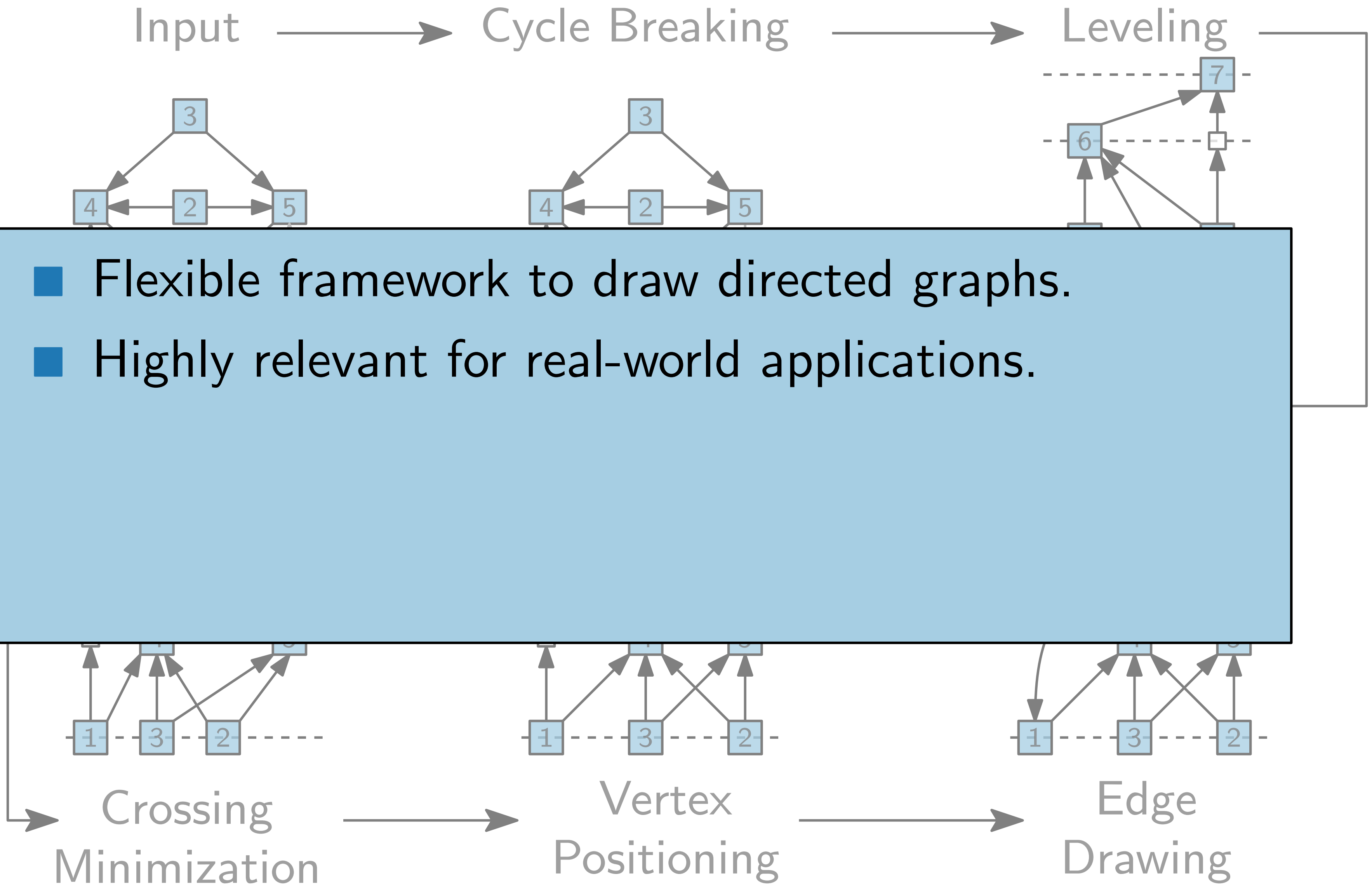
[Sugiyama, Tagawa, Toda ’81]



- Flexible framework to draw directed graphs.

Classical Approach – Sugiyama Framework

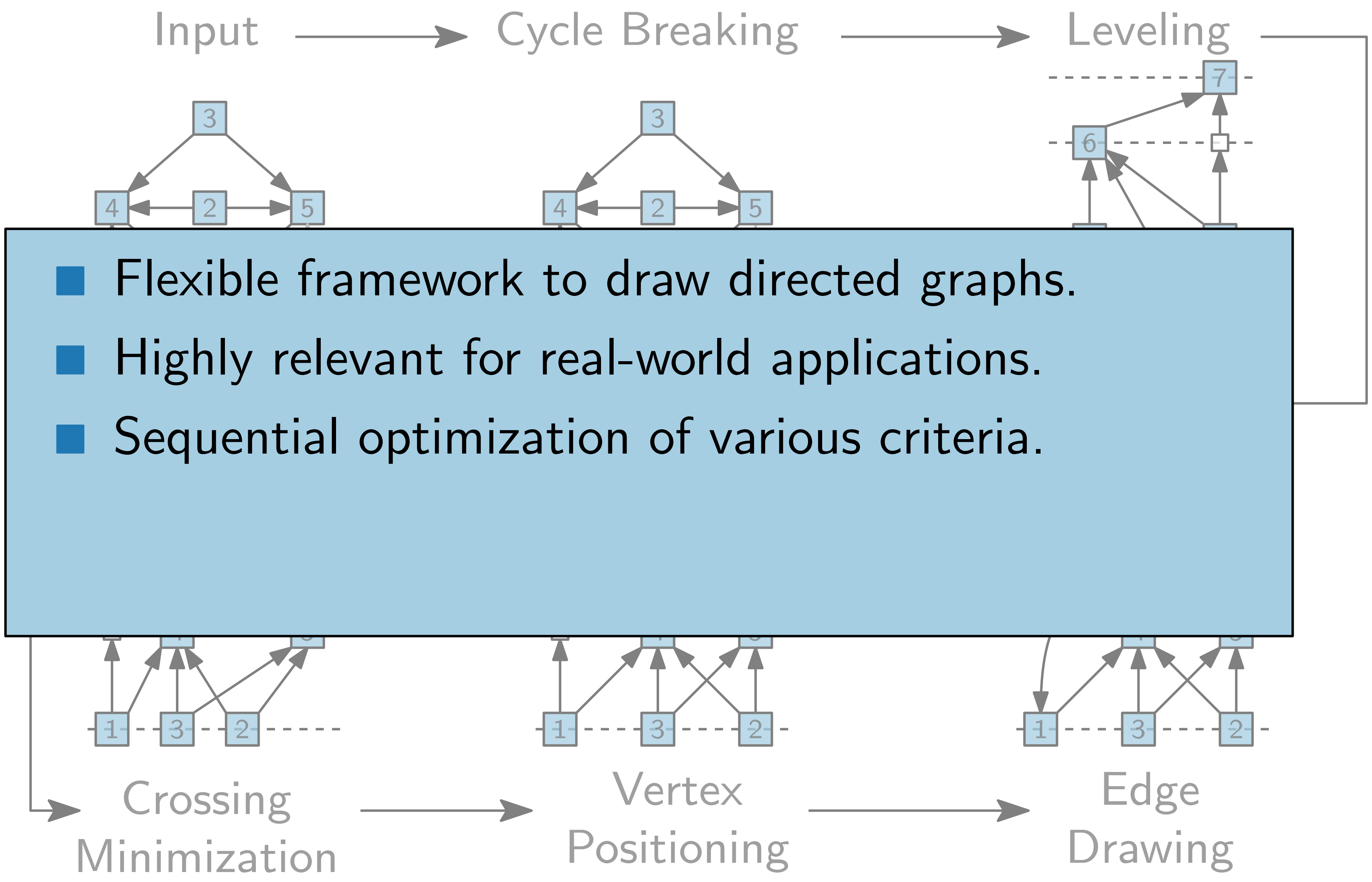
[Sugiyama, Tagawa, Toda '81]



- Flexible framework to draw directed graphs.
- Highly relevant for real-world applications.

Classical Approach – Sugiyama Framework

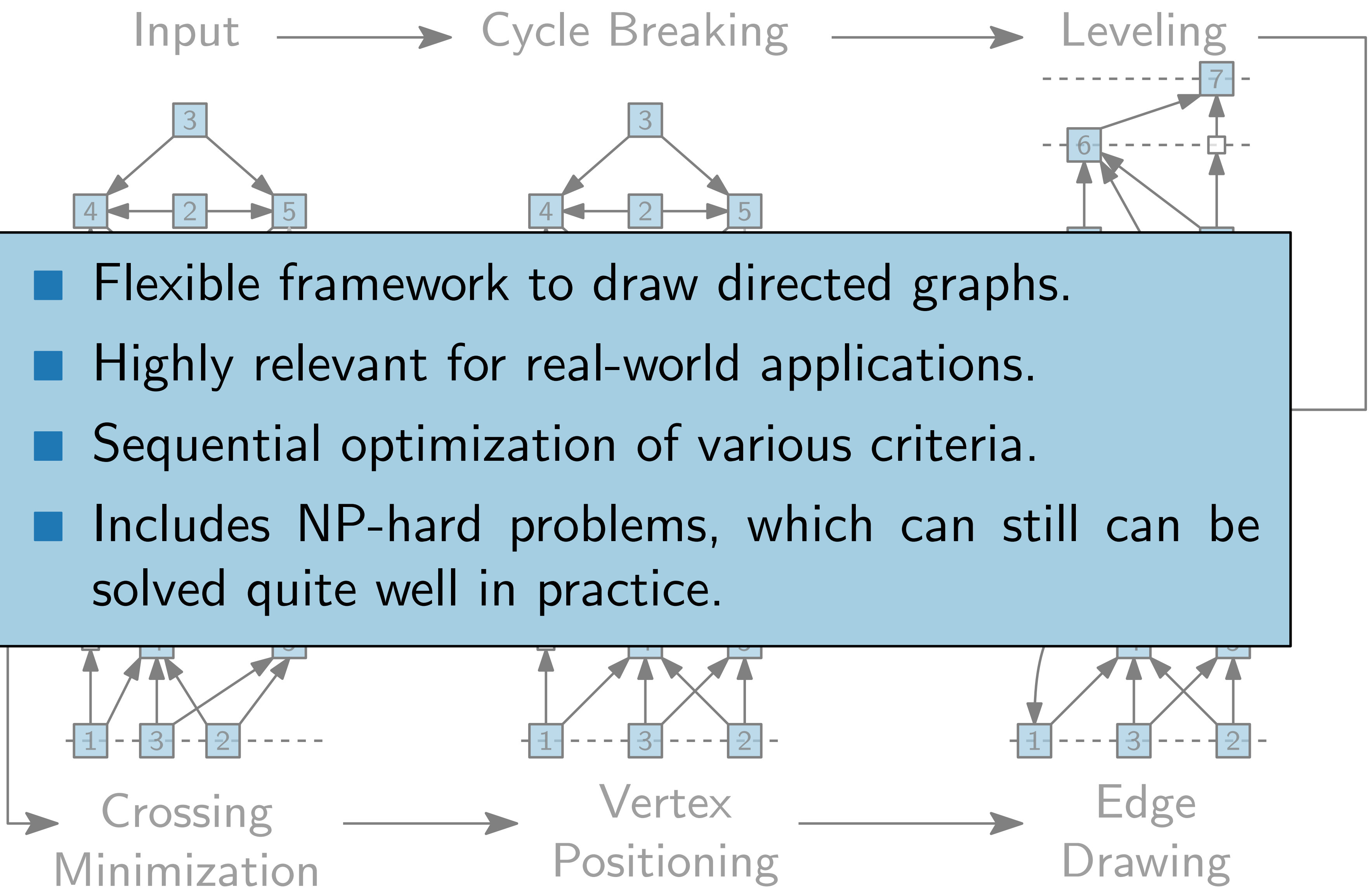
[Sugiyama, Tagawa, Toda '81]



- Flexible framework to draw directed graphs.
- Highly relevant for real-world applications.
- Sequential optimization of various criteria.

Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]



- Flexible framework to draw directed graphs.
- Highly relevant for real-world applications.
- Sequential optimization of various criteria.
- Includes NP-hard problems, which can still can be solved quite well in practice.

Literature

Detailed explanations of steps and proofs in

- [GD Ch. 11] and [DG Ch. 5]

based on

- [Sugiyama, Tagawa, Toda '81] Methods for visual understanding of hierarchical system structures

and refined with results from

- [Berger, Shor '90] Approximation algorithms for the maximum acyclic subgraph problem
- [Eades, Lin, Smith '93] A fast and effective heuristic for the feedback arc set problem
- [Garey, Johnson '83] Crossing number is NP-complete
- [Eades, Kelly '86] Heuristics for reducing crossings in 2-layered networks.
- [Eades, Whiteside '94] Drawing graphs in two layers
- [Eades, Wormland '94] Edge crossings in drawings of bipartite graphs
- [Jünger, Mutzel '97] 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms