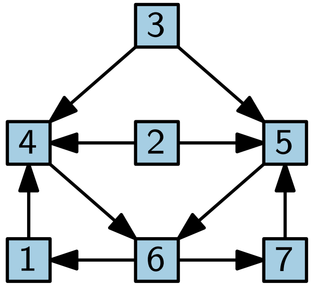
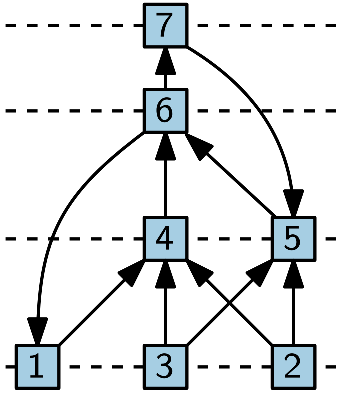
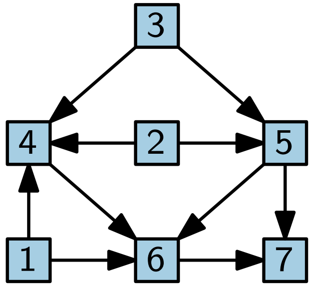
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

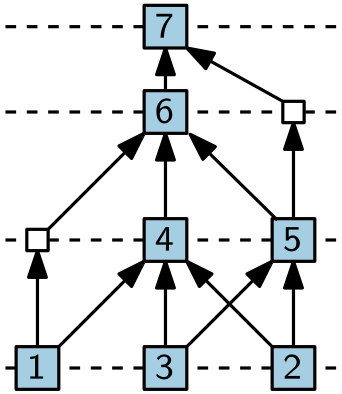
Visualization of Graphs

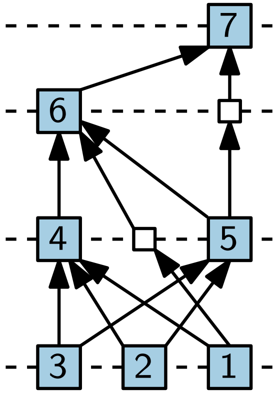




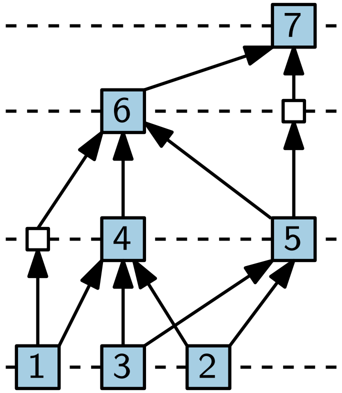
Lecture 8:
Hierarchical Layouts:
Sugiyama Framework





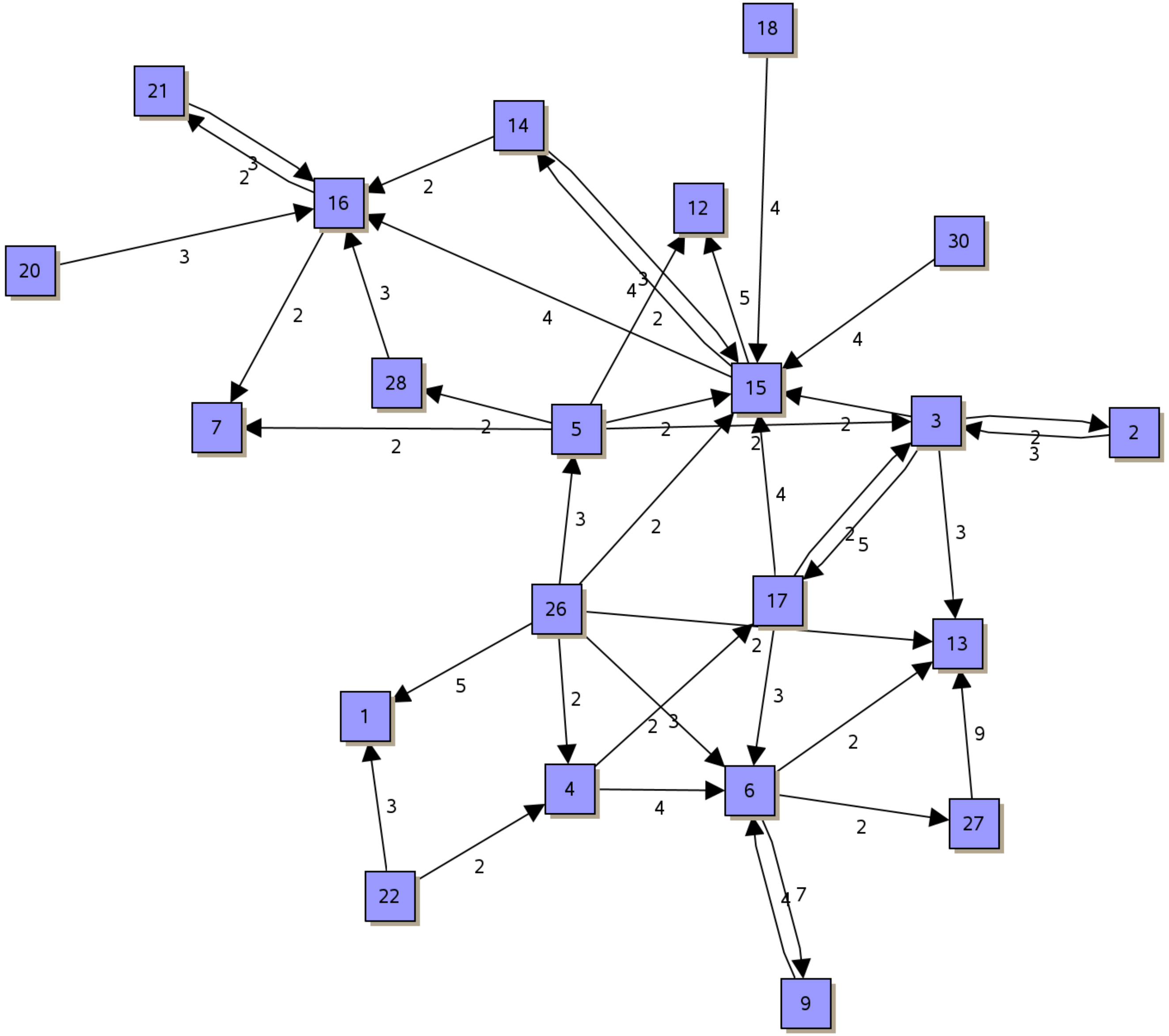


Alexander Wolff

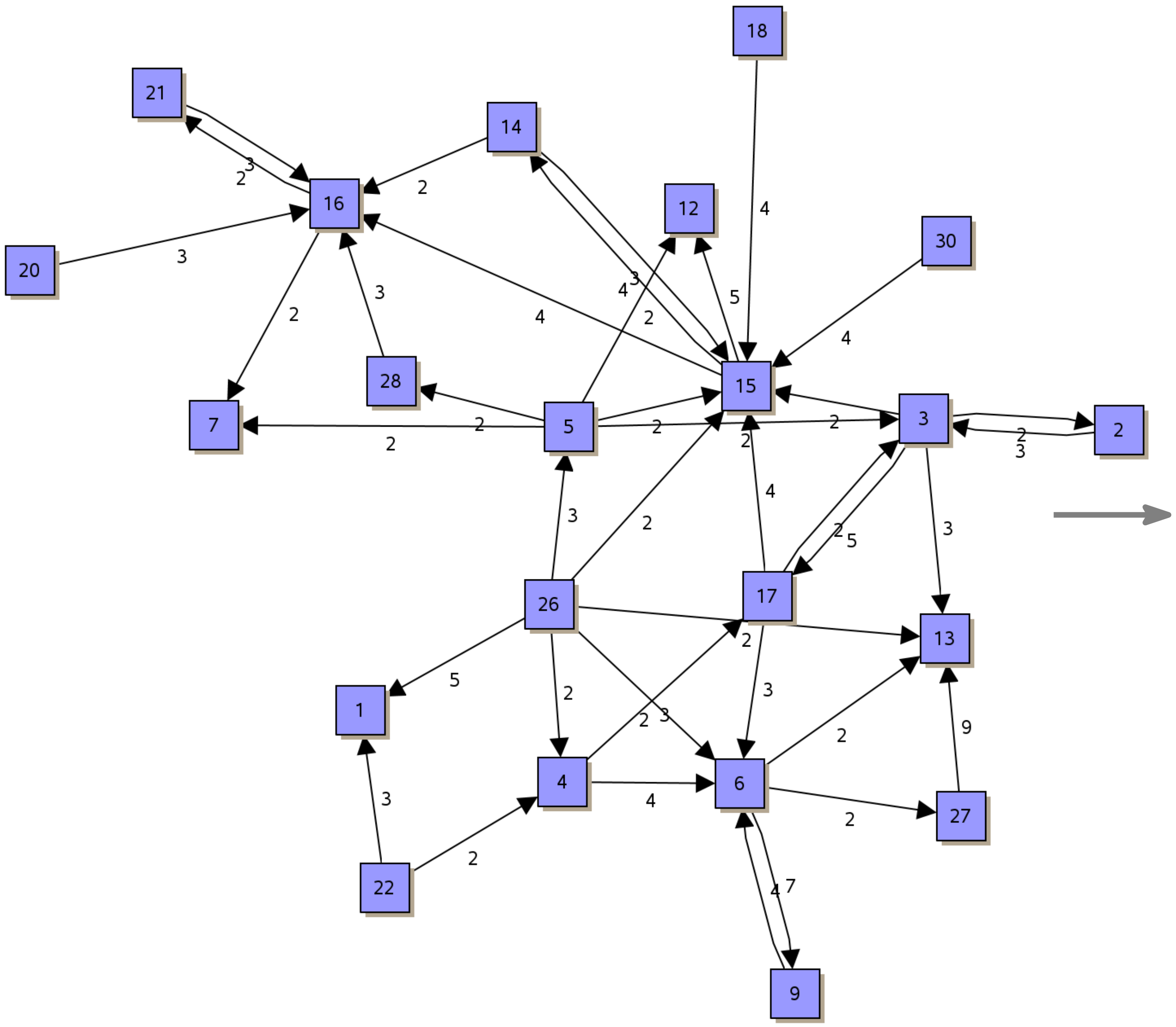


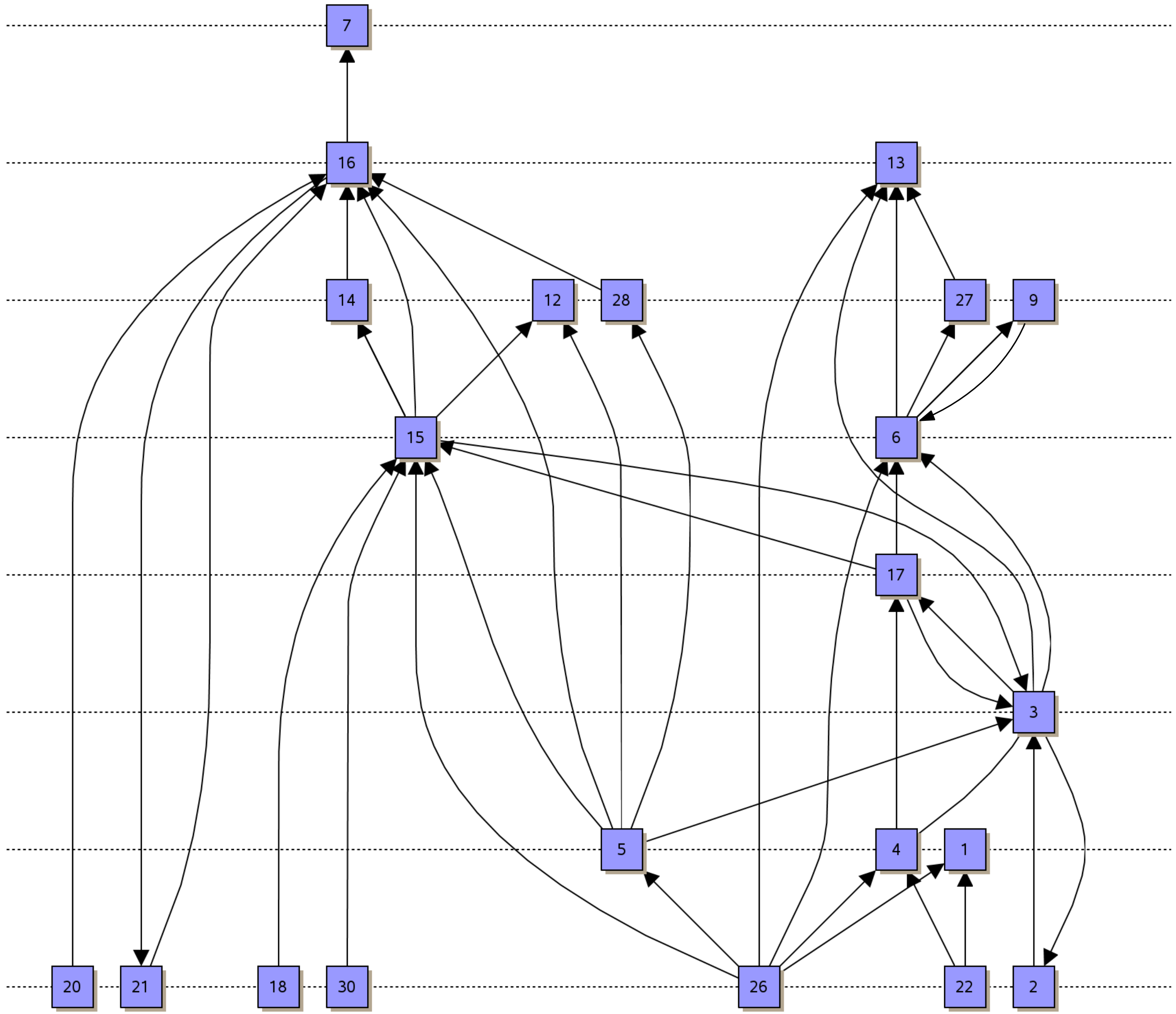
Summer term 2025

Hierarchical Drawings – Motivation



Hierarchical Drawings – Motivation

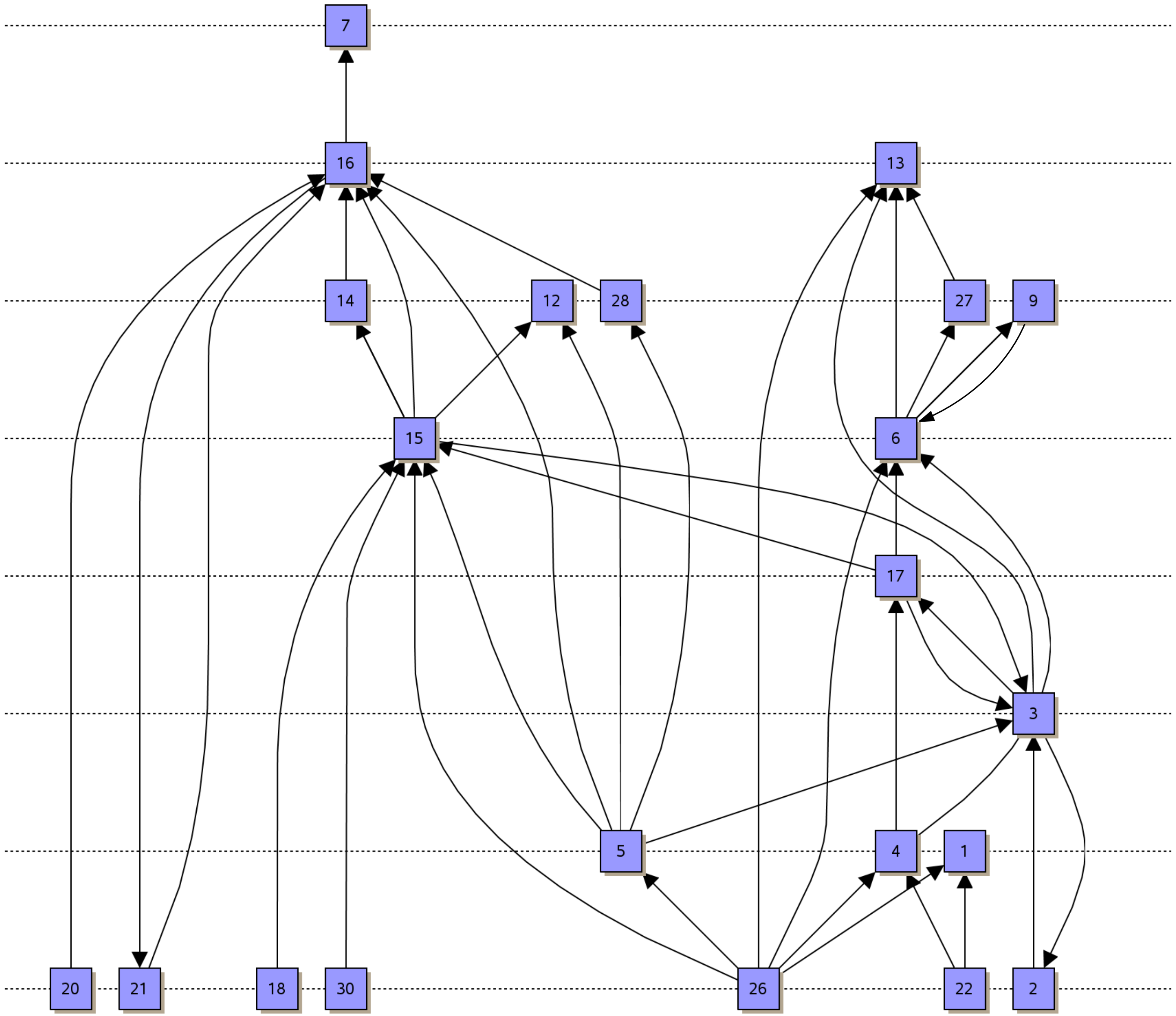




Hierarchical Drawing

Problem Statement:

- Input: digraph G
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

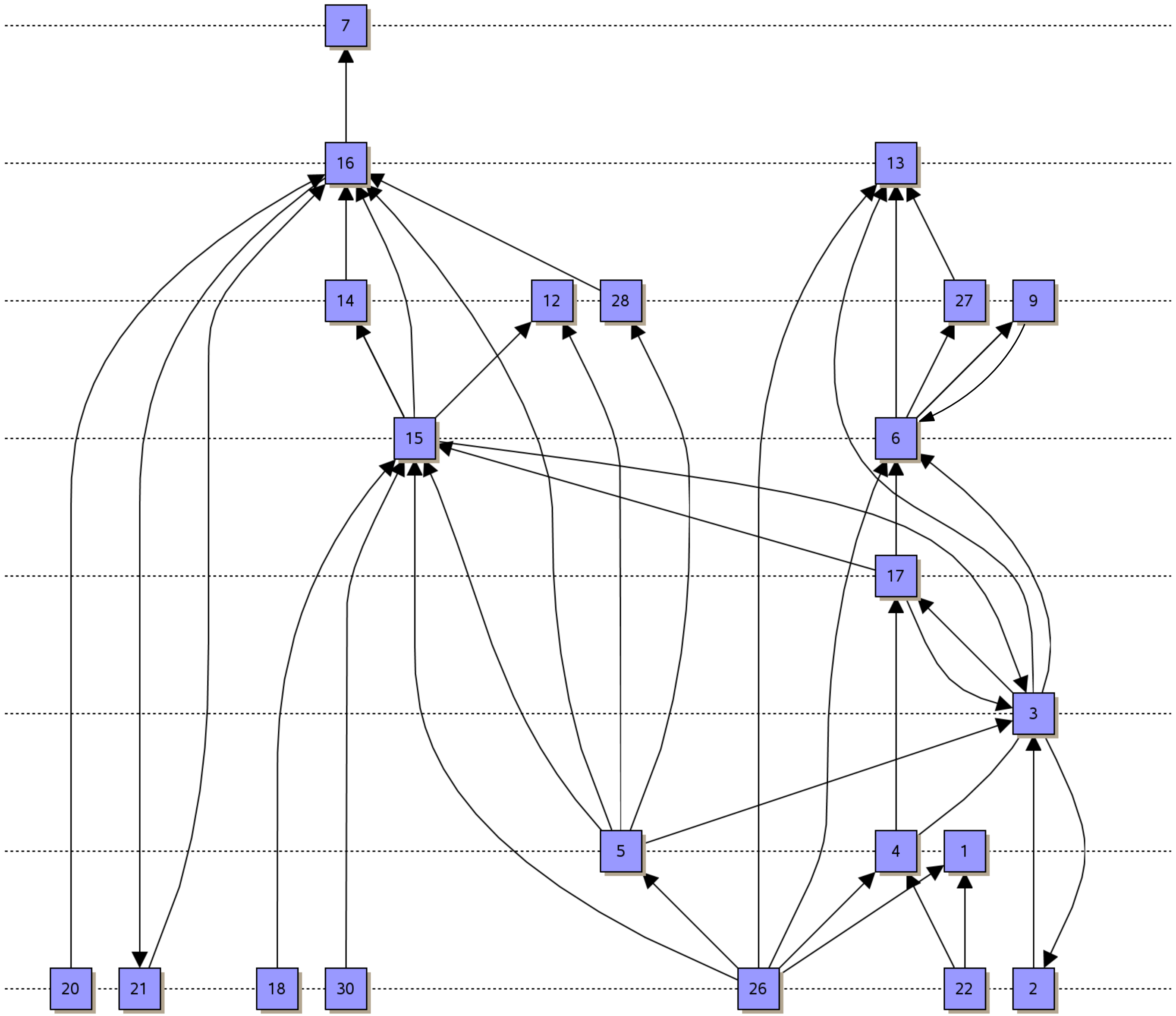


Hierarchical Drawing

Problem Statement:

- Input: digraph G
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties:



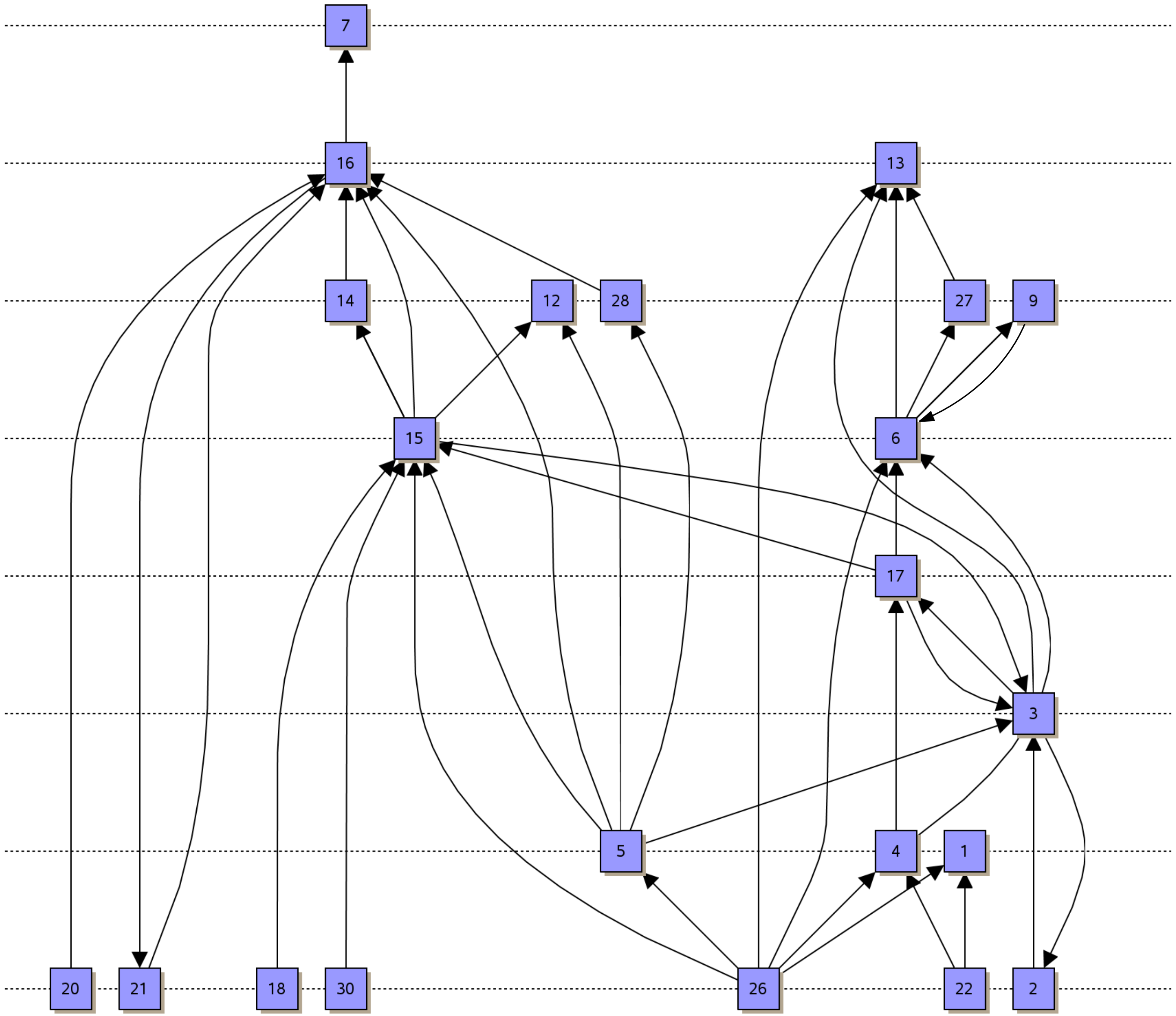
Hierarchical Drawing

Problem Statement:

- Input: digraph G
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties:

- edges are directed upwards,



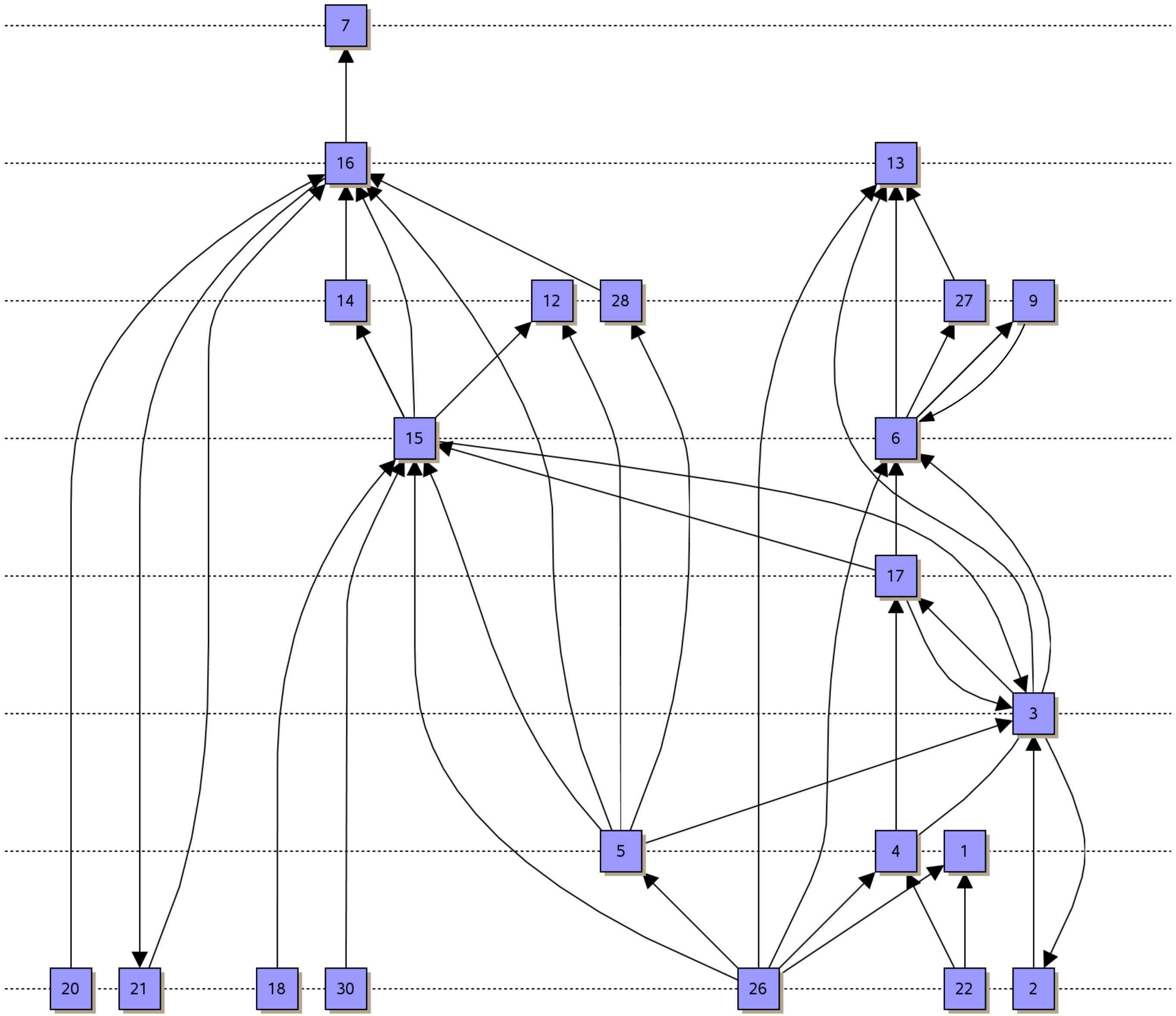
Hierarchical Drawing

Problem Statement:

- Input: digraph G
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties:

- edges are directed upwards,
- vertices lie on (few) horizontal lines,



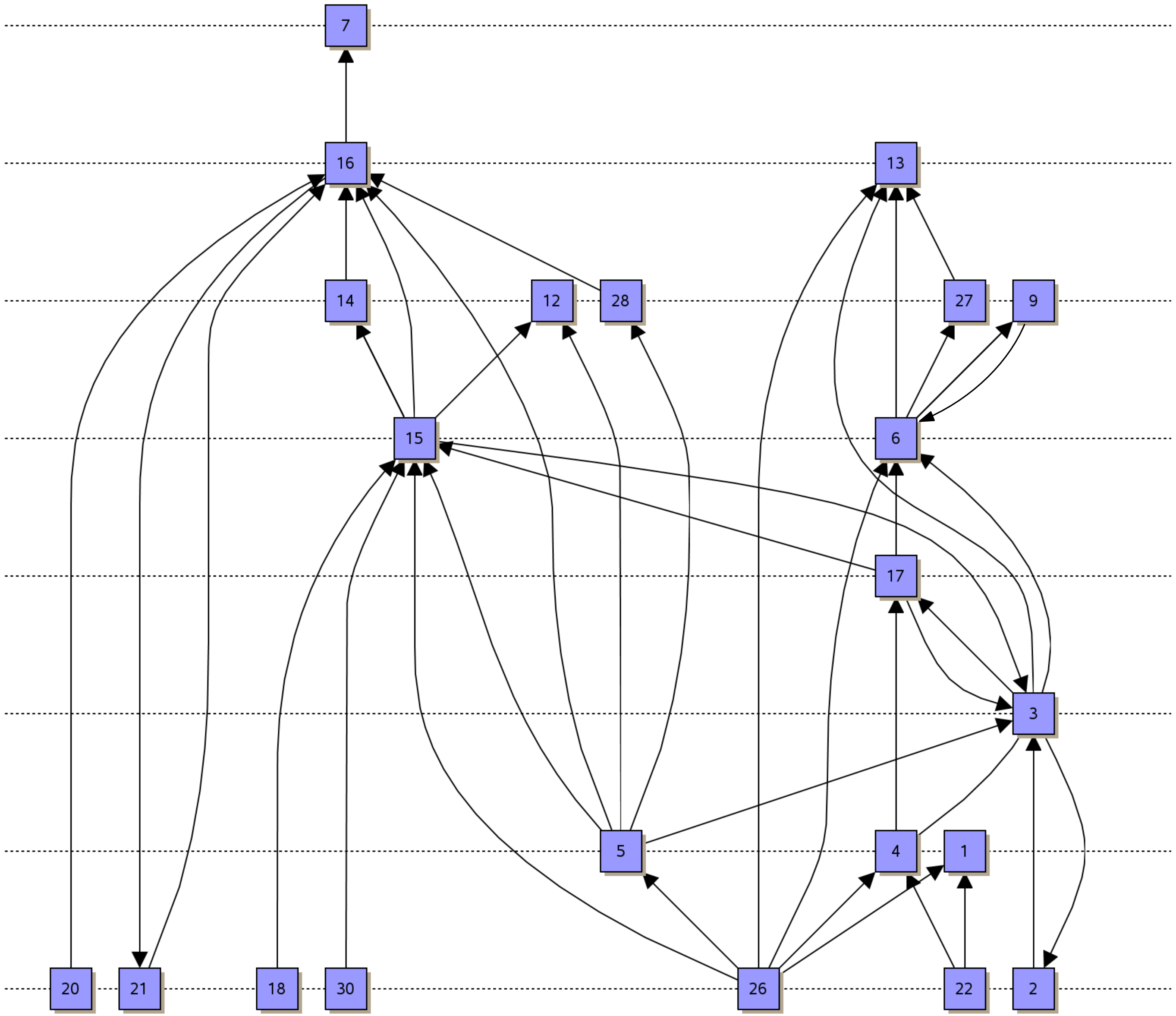
Hierarchical Drawing

Problem Statement:

- Input: digraph G
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties:

- edges are directed upwards,
- vertices lie on (few) horizontal lines,
- few pairs of edges cross,



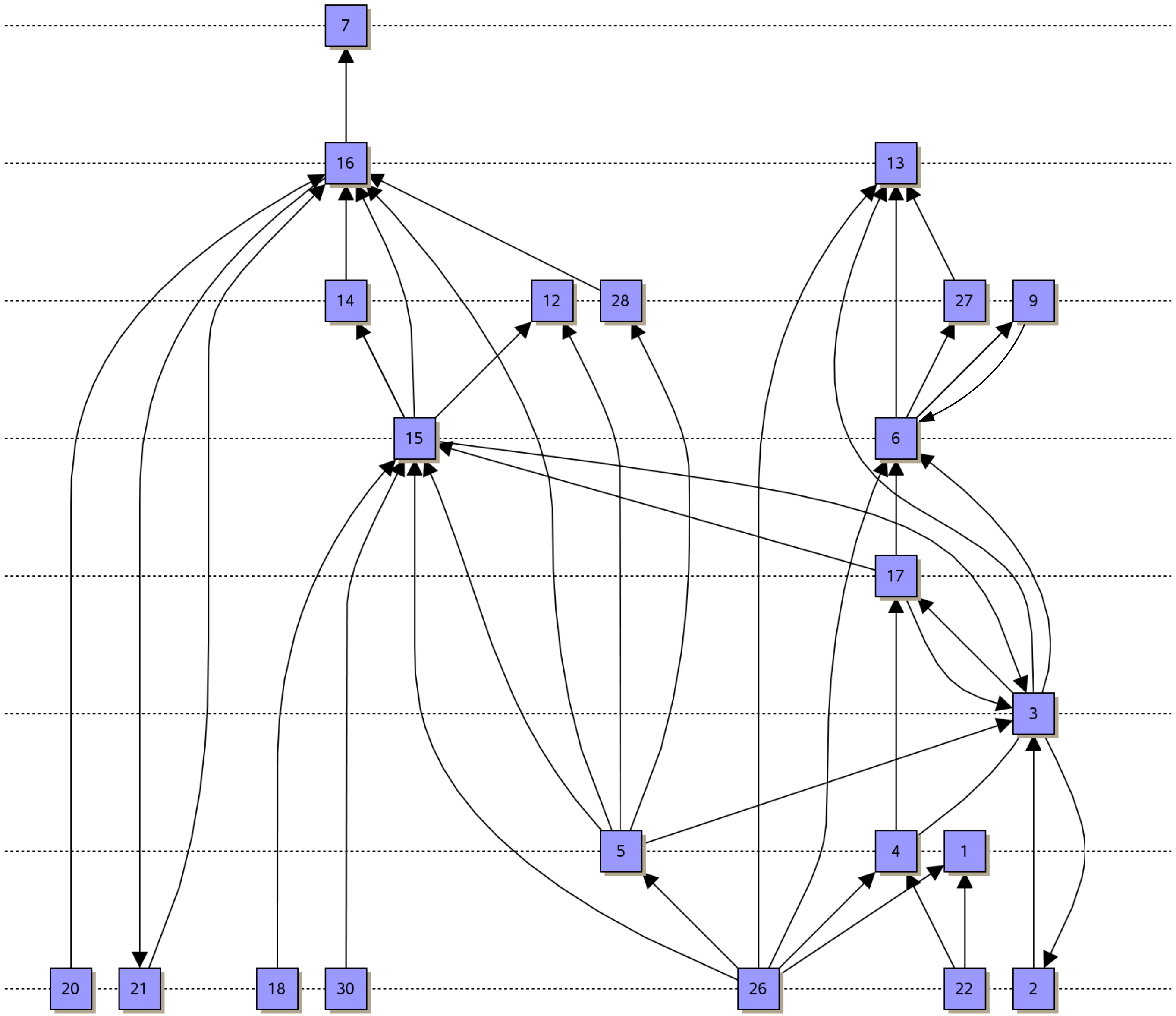
Hierarchical Drawing

Problem Statement:

- Input: digraph G
- Output: drawing of G that "closely" reproduces the hierarchical properties of G

Desirable Properties:

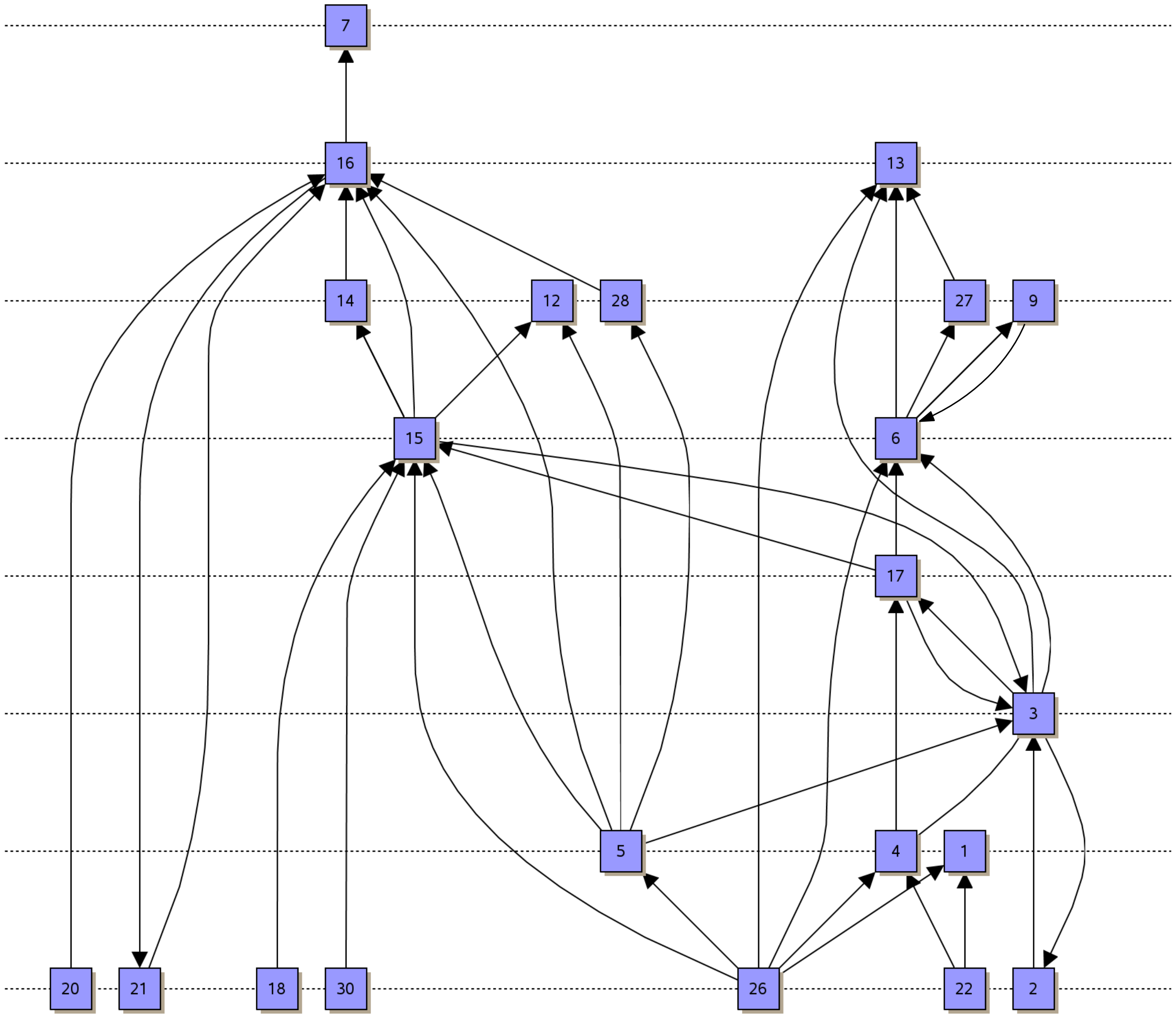
- edges are directed upwards,
- vertices lie on (few) horizontal lines,
- few pairs of edges cross,
- edges are short,



Hierarchical Drawing

Problem Statement:

- Input: digraph G
- Output: drawing of G that "closely" reproduces the hierarchical properties of G



Desirable Properties:

- edges are directed upwards,
- vertices lie on (few) horizontal lines,
- few pairs of edges cross,
- edges are short,
- vertices are evenly spaced.

Hierarchical Drawing

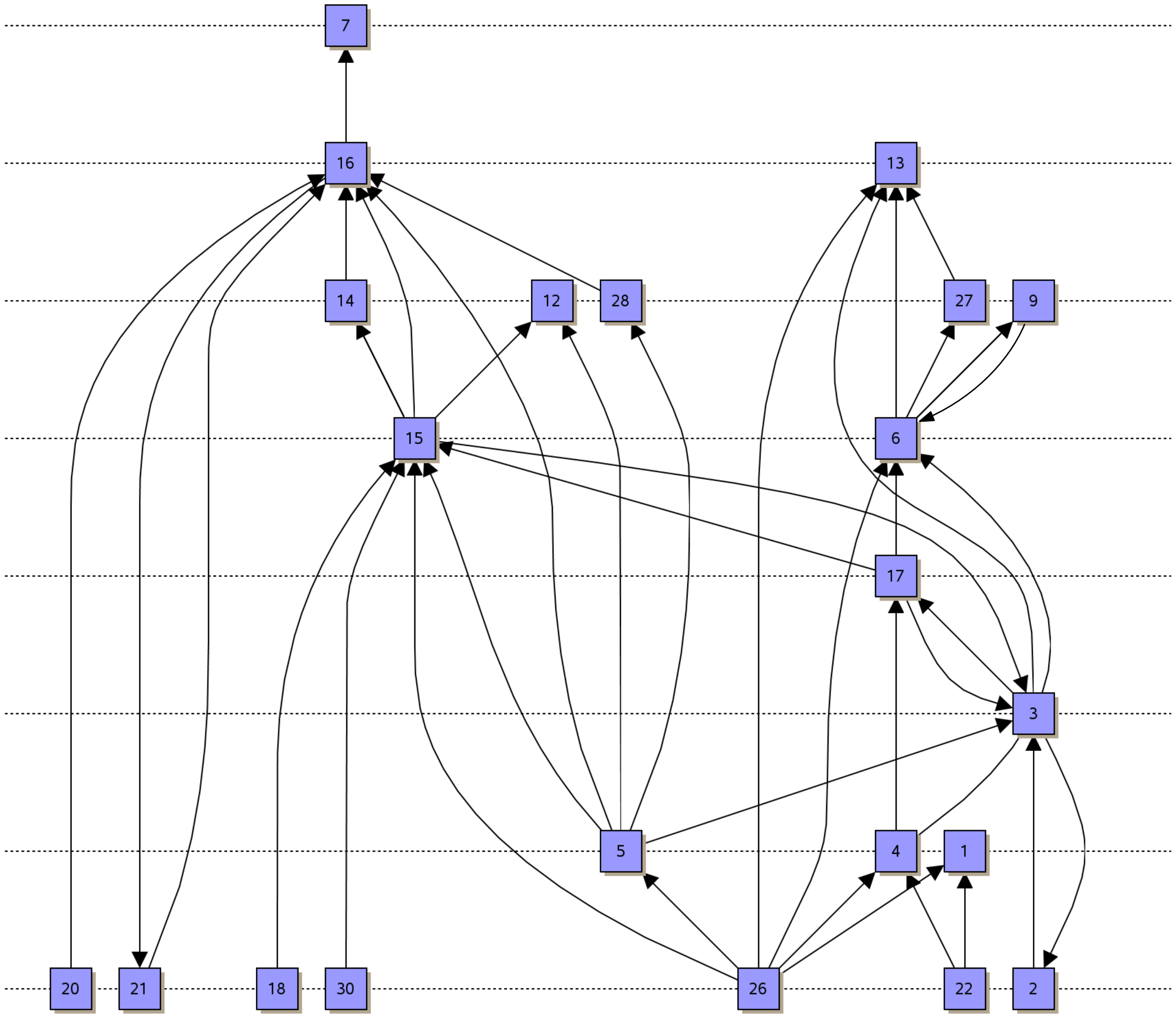
Problem Statement:

- Input: digraph G
- Output: drawing of G that “closely” reproduces the hierarchical properties of G

Desirable Properties:

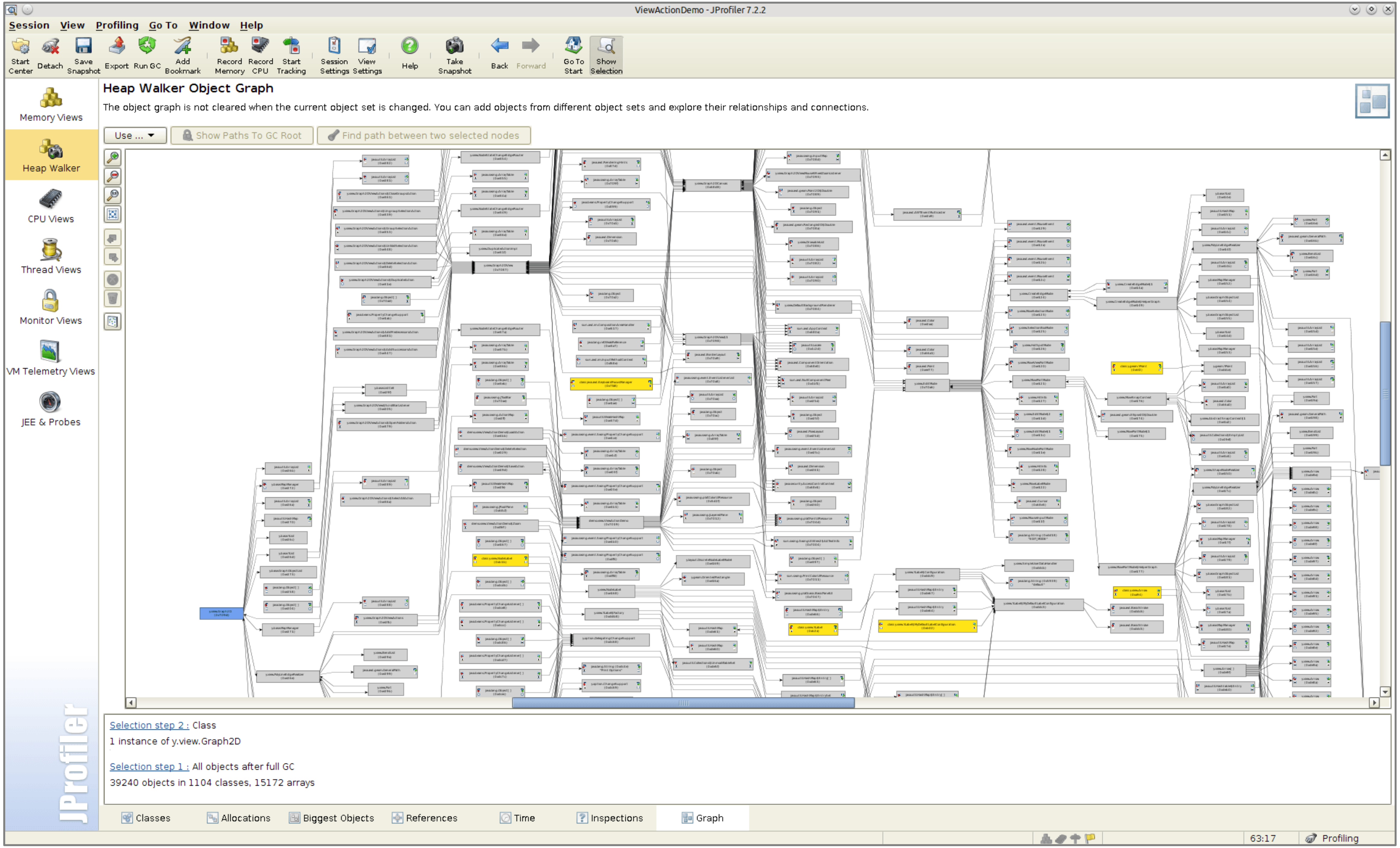
- edges are directed upwards,
- vertices lie on (few) horizontal lines,
- few pairs of edges cross,
- edges are short,
- vertices are evenly spaced.

Criteria can be contradictory!

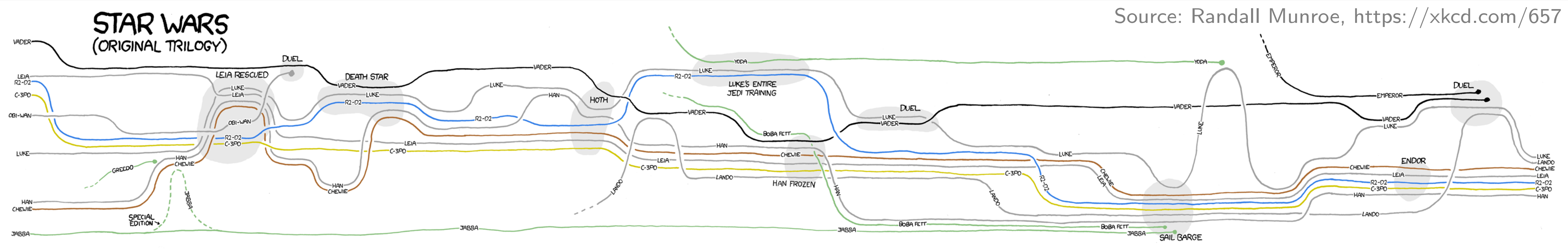


Hierarchical Drawing – Applications

yEd Gallery: Java profiler JProfiler using yFiles

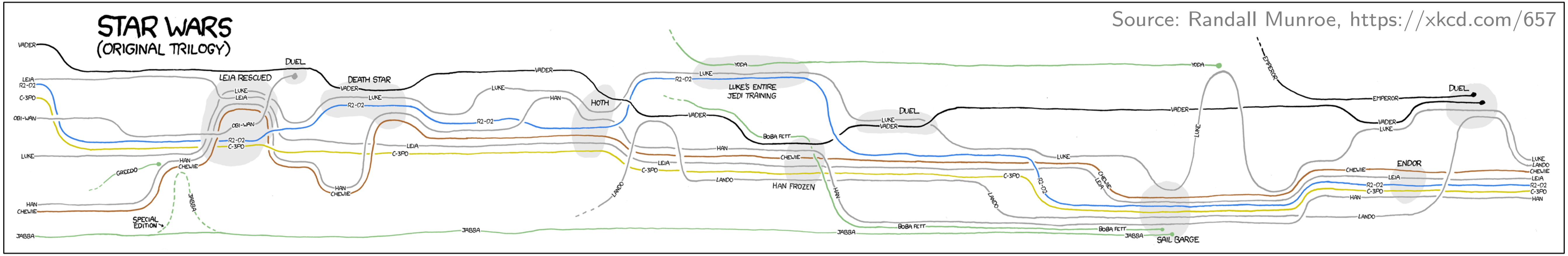


Hierarchical Drawing – Applications

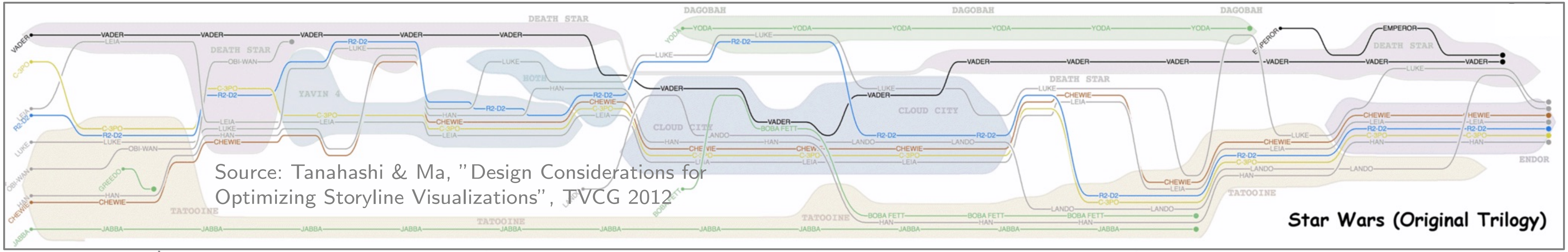


Source: Randall Munroe, https://xkcd.com/657

Hierarchical Drawing – Applications

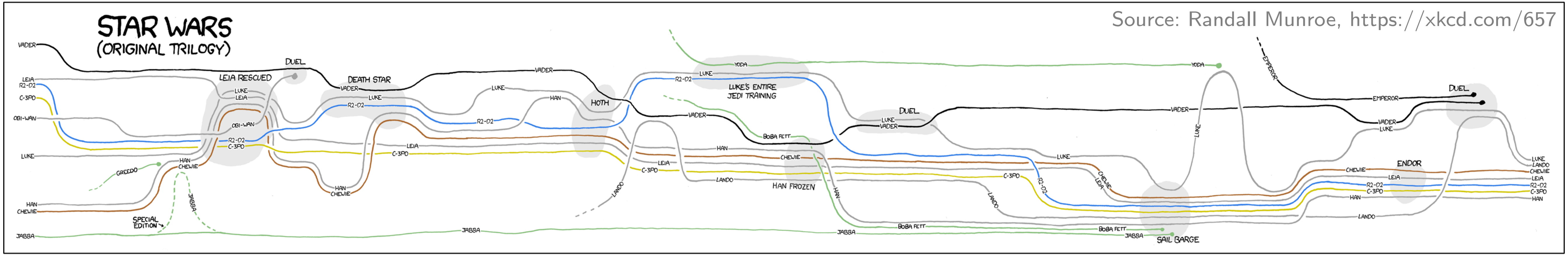


Source: Randall Munroe, https://xkcd.com/657



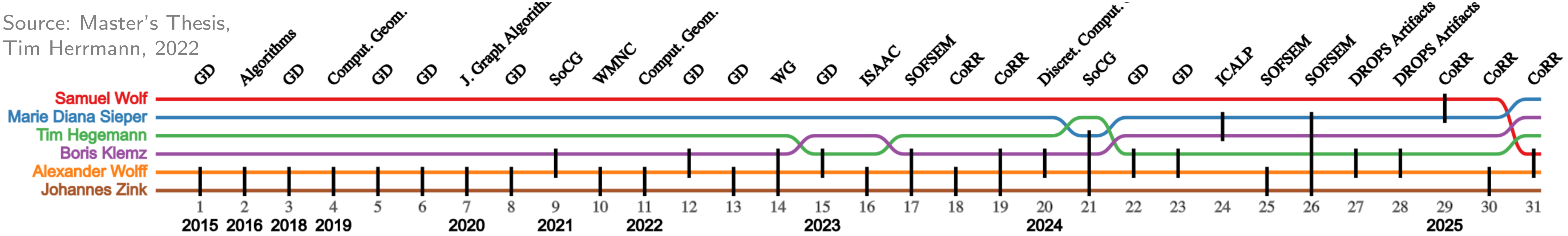
Source: Tanahashi & Ma, "Design Considerations for Optimizing Storyline Visualizations", TVCG 2012

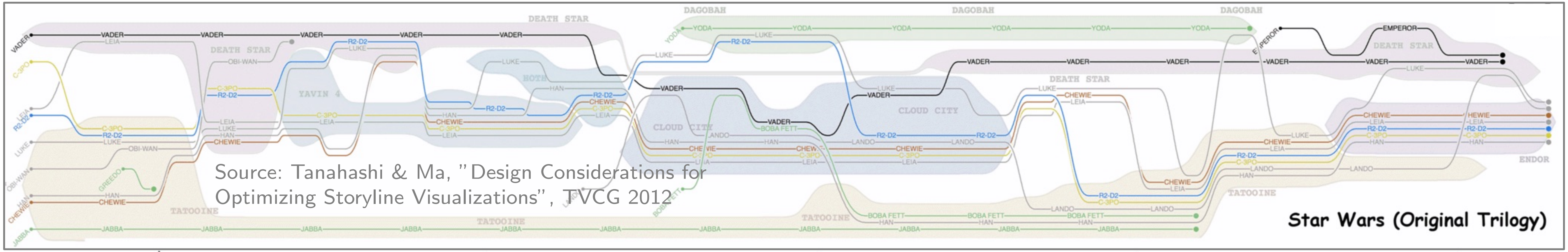
Hierarchical Drawing – Applications



Source: Randall Munroe, https://xkcd.com/657

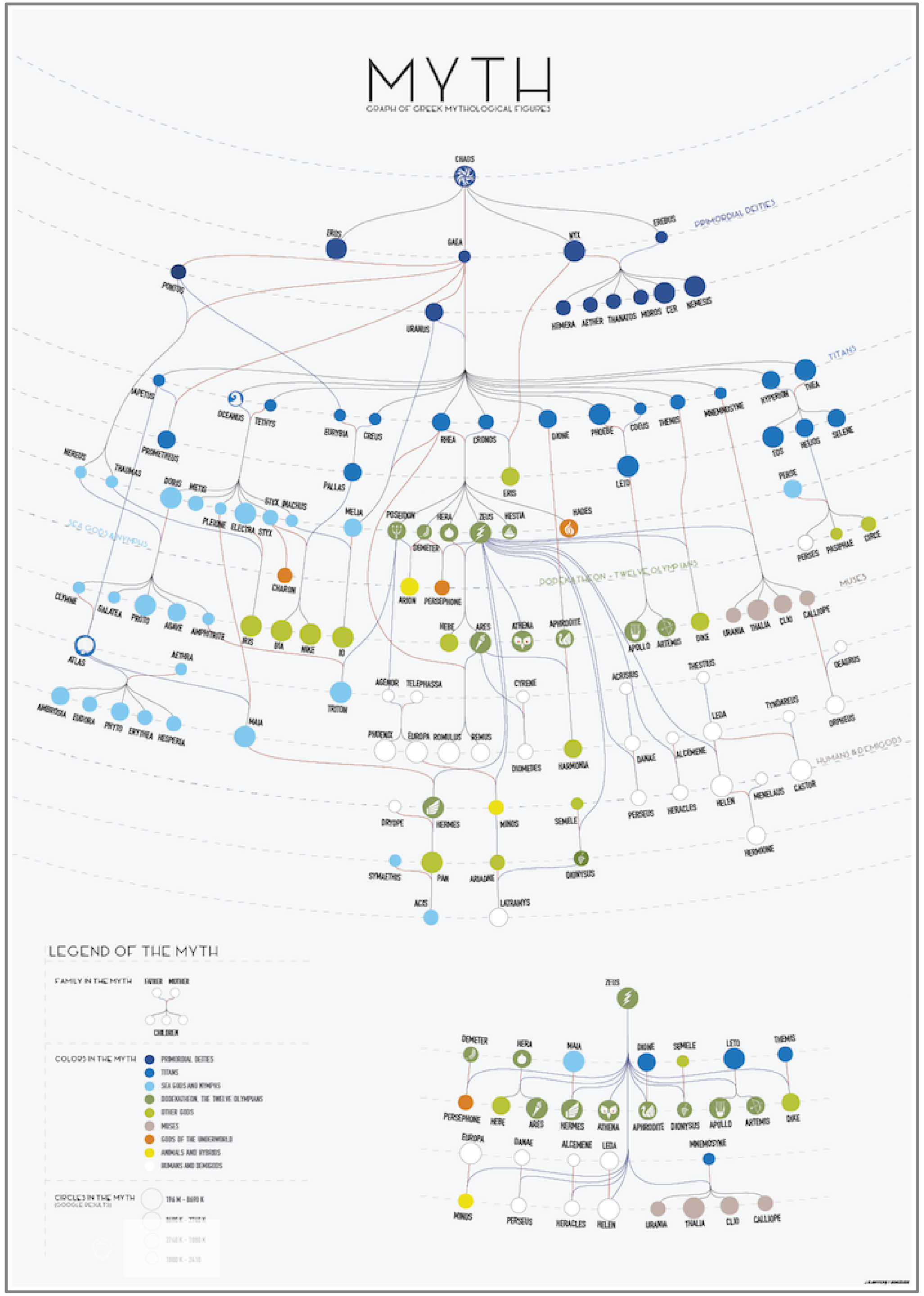
Source: Master's Thesis,
Tim Herrmann, 2022





Source: Tanahashi & Ma, "Design Considerations for Optimizing Storyline Visualizations", TVCG 2012

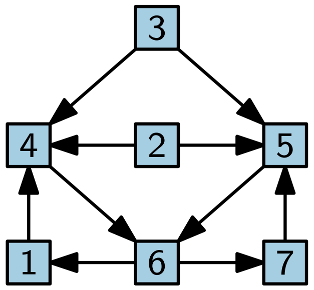
Hierarchical Drawing – Applications



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

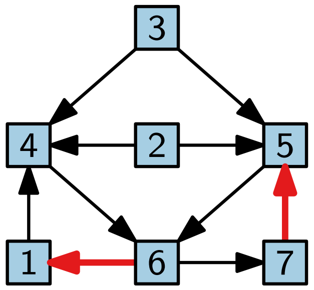
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda '81]

Input



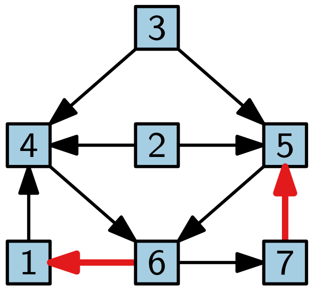
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda '81]

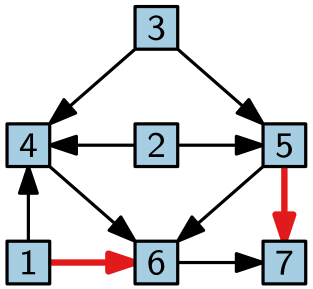
Input



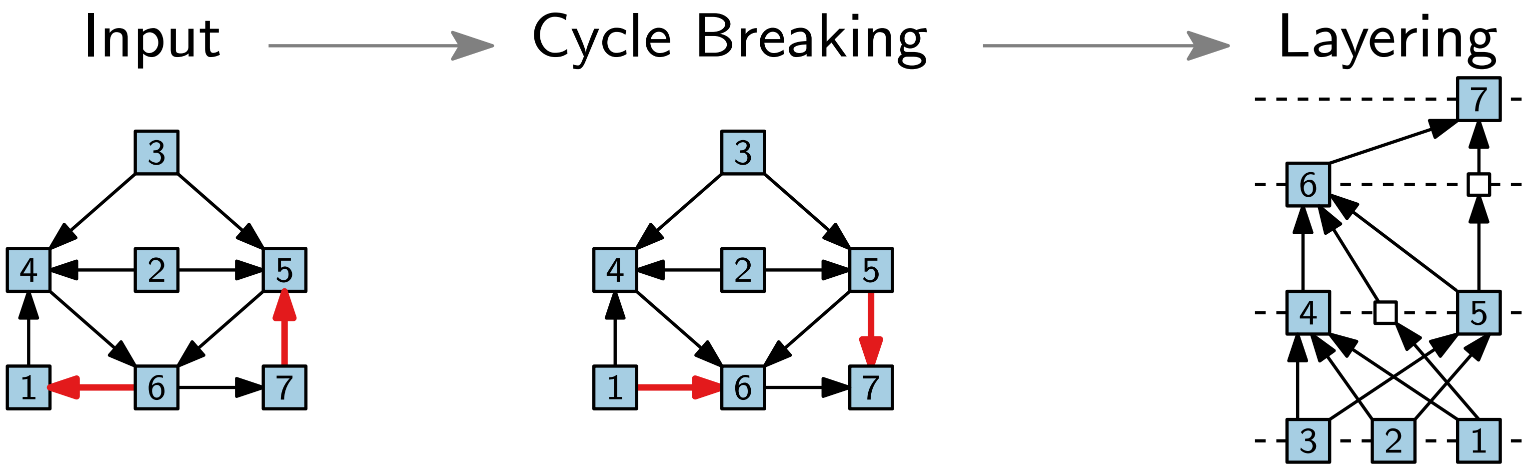
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda '81]

Input → Cycle Breaking

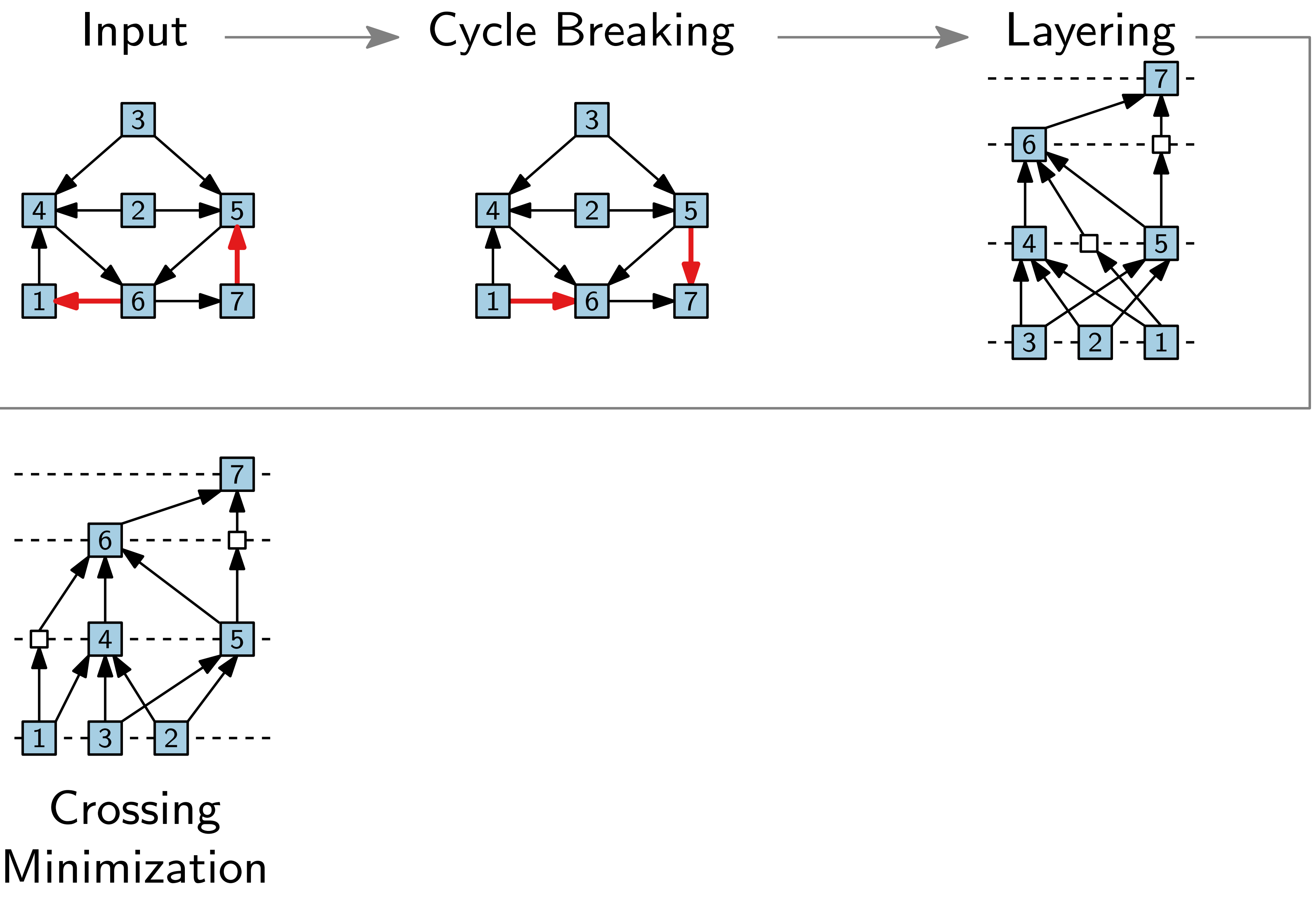




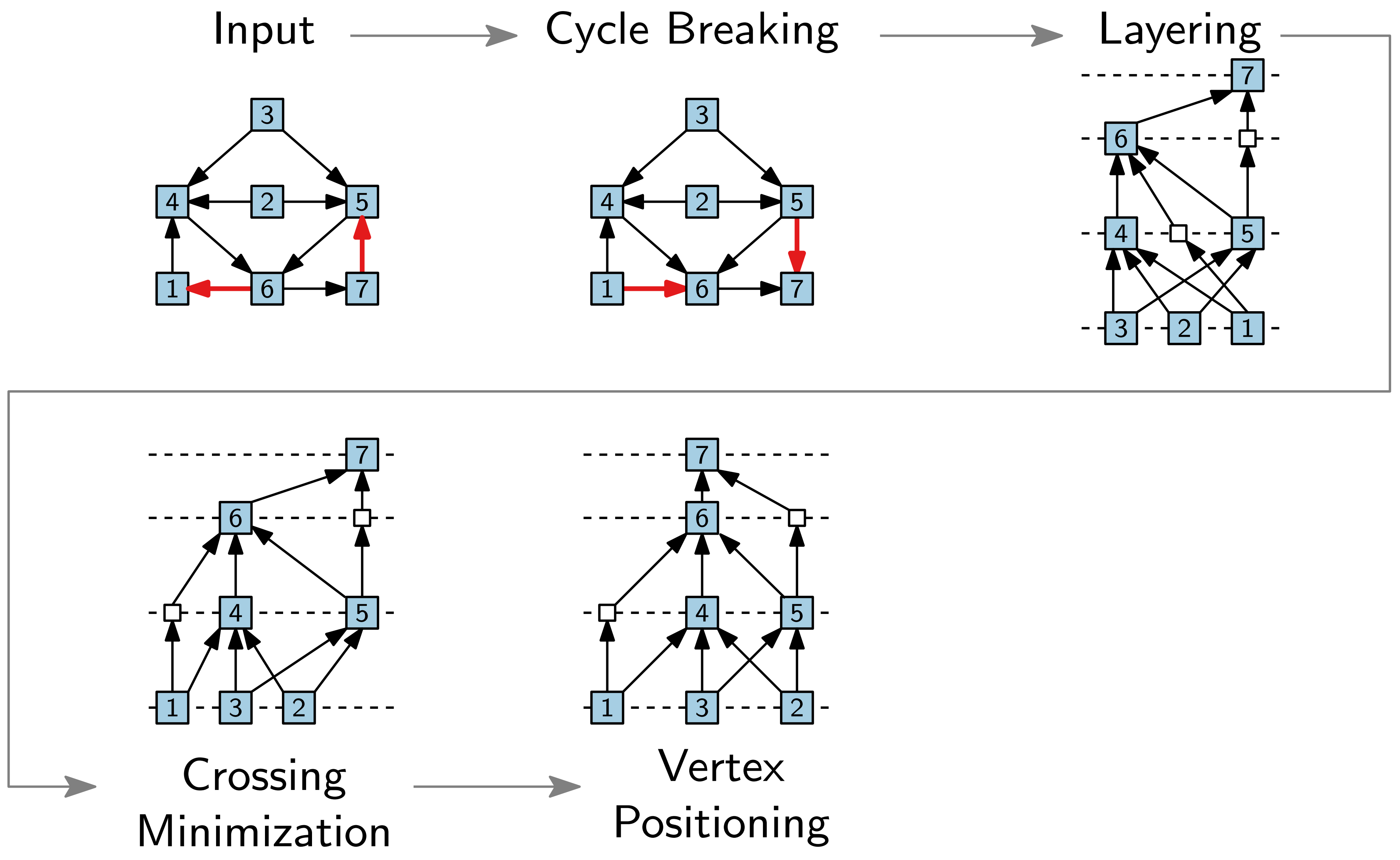
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda ’81]



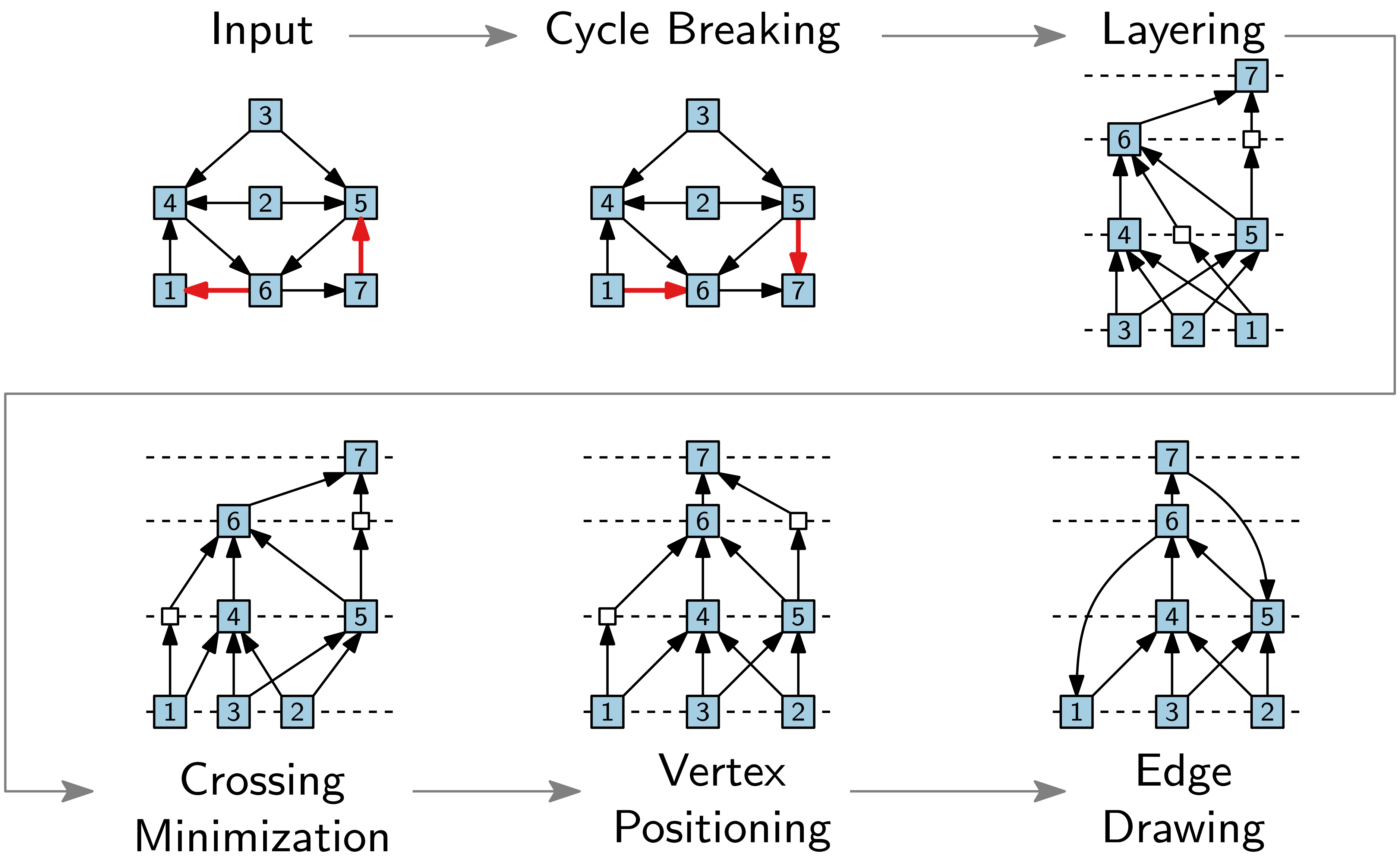
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda ’81]



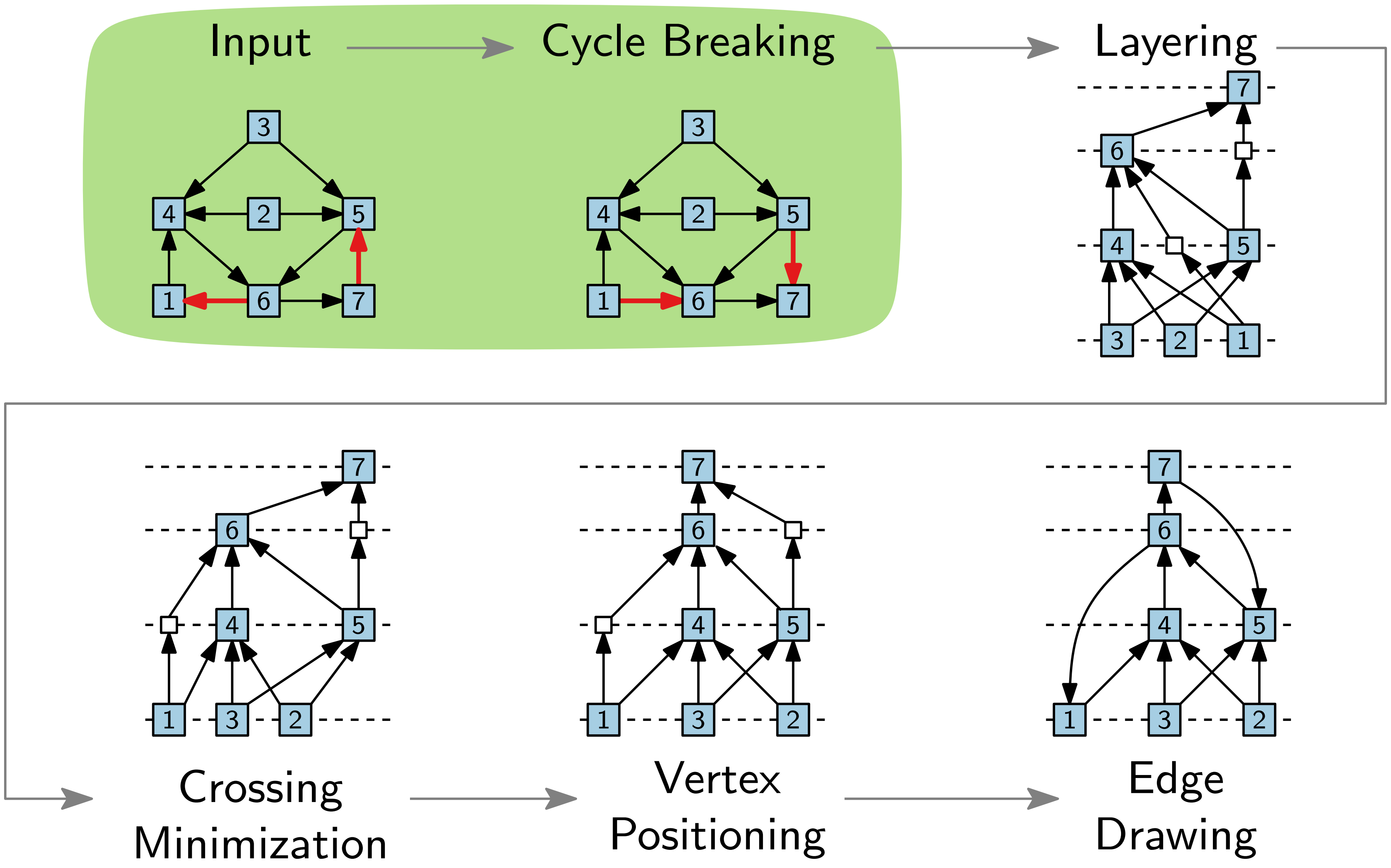
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda '81]



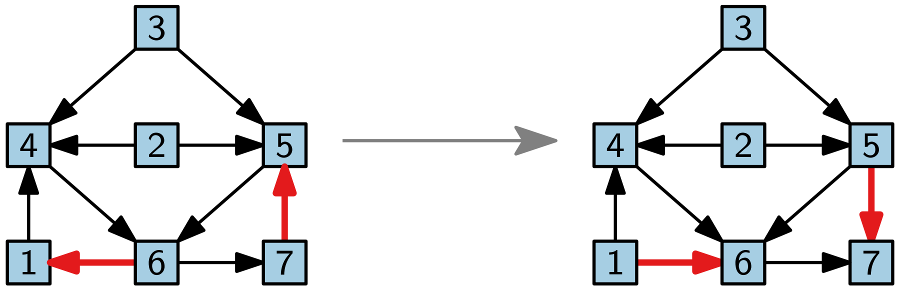
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda '81]



Step 1: Cycle Breaking

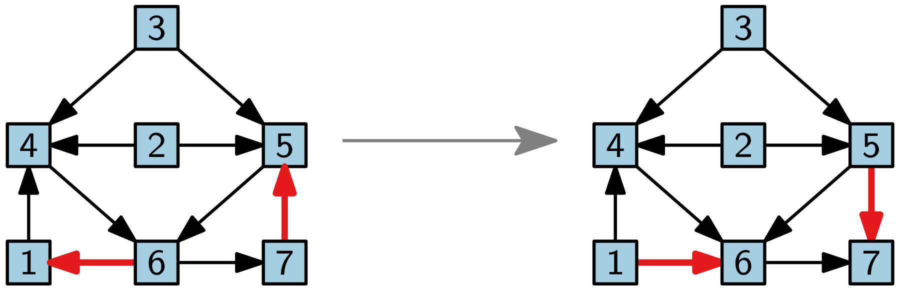


Step 1: Cycle Breaking



Approach.

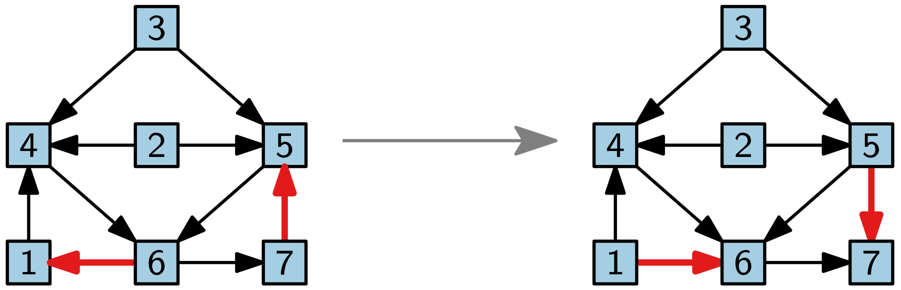
Step 1: Cycle Breaking



Approach.

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

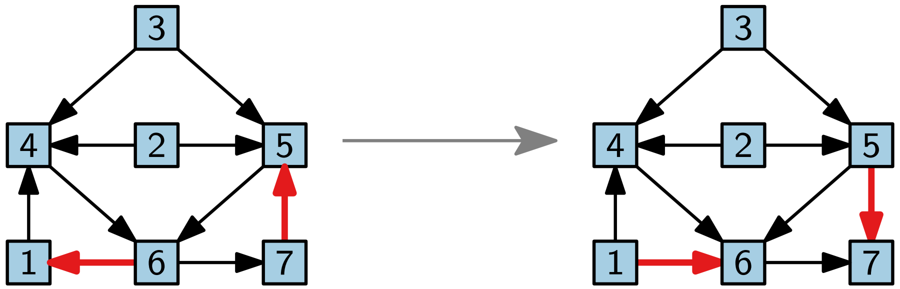
Step 1: Cycle Breaking



Approach.

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

Step 1: Cycle Breaking

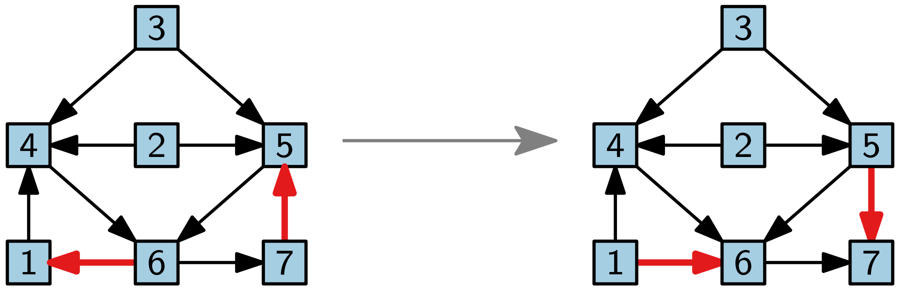


Approach.

- 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



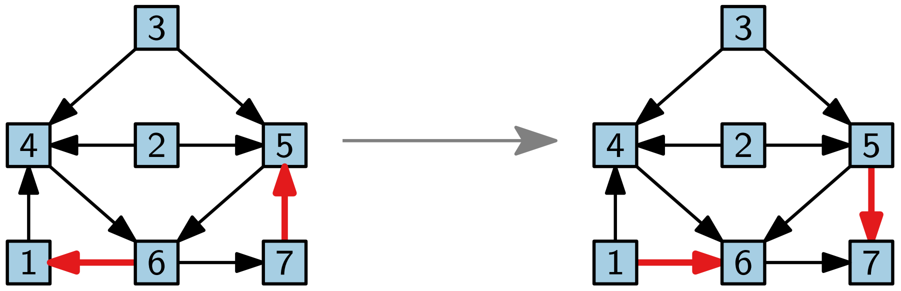
Approach.

- 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
- Output:

Step 1: Cycle Breaking



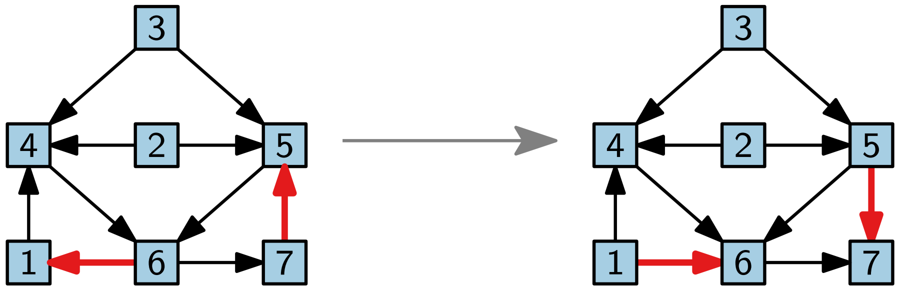
Approach.

- 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
- Output: minimum-size set $E^\star \subseteq E(G)$ such that $G^\star = (V(G), E(G) \setminus E^\star)$ acyclic

Step 1: Cycle Breaking



Approach.

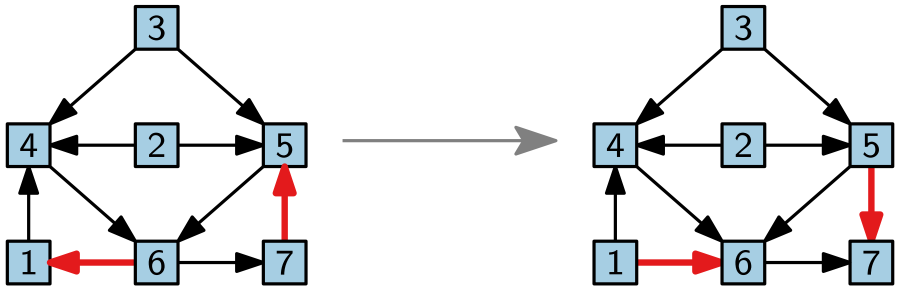
- Find minimum-size set $E^\star$ of edges that are not upward.
- Remove $E^\star$ and insert reversed edges.

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

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

$$(E(G) \setminus E^\star) \cup E^\star_{\text{rev}}$$

Step 1: Cycle Breaking



Approach.

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

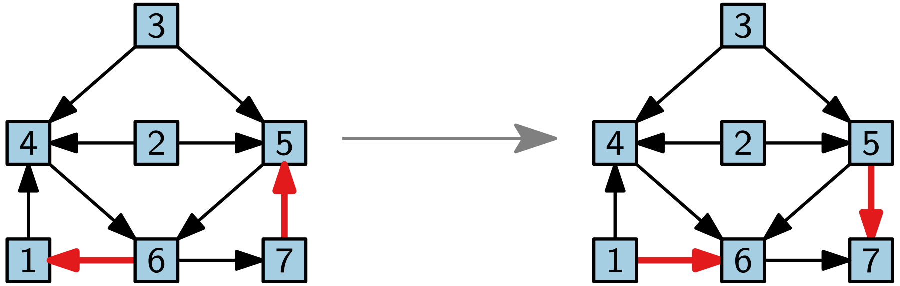
Problem Minimum Feedback ~~Arc~~ Set **(F~~A~~S).**

edges in $E^\star$ but reversed

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

$(E(G) \setminus E^\star) \cup \boxed{E^\star_{\text{rev}}}$

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 $\textsc{Minimum Feedback}$ ~~$\textsc{Arc}$~~ $\textsc{Set}$ **(F~~A~~S).**

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

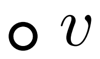
edges in $E^\star$ but reversed

$(E(G) \setminus E^\star) \cup E^\star_{\text{rev}}$

. . . NP-hard ☹

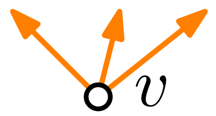
Heuristic 1

[Berger, Shor '90]



Heuristic 1

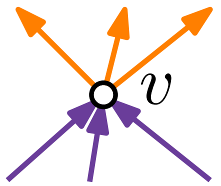
[Berger, Shor '90]



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

Heuristic 1

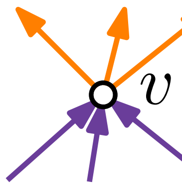
[Berger, Shor '90]



$$E^{\rightarrow}(v) \quad := \quad \{(v,u)\colon (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) \quad := \quad \{(u,v)\colon (u,v) \in E(G)\}$$

Heuristic 1

[Berger, Shor '90]

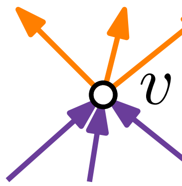


$$
\begin{aligned}
E^{\rightarrow}(v) &:= \{(v,u)\colon (v,u) \in E(G)\} \\
E^{\leftarrow}(v) &:= \{(u,v)\colon (u,v) \in E(G)\} \\
E(v) &:= E^{\rightarrow}(v) \cup E^{\leftarrow}(v)
\end{aligned}
$$

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):



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

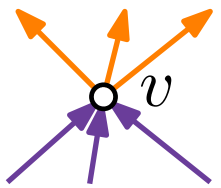
Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

return $G' = (V(G), E')$



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

Heuristic 1

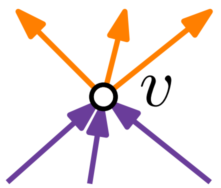
[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

return $G' = (V(G), E')$



$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$

$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$

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

Heuristic 1

[Berger, Shor '90]

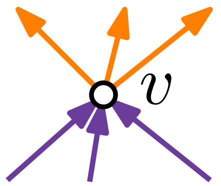
GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

return $G' = (V(G), E')$



$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$

$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$

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

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

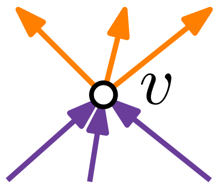
$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

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

return $G' = (V(G), E')$



$$E^{\rightarrow}(v) := \{(v,u)\colon (v,u) \in E(G)\}$$

$$E^{\leftarrow}(v) := \{(u,v)\colon (u,v) \in E(G)\}$$

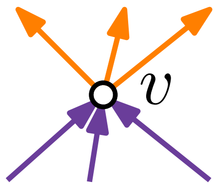
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

Heuristic 1

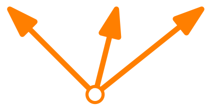
[Berger, Shor '90]

```
GreedyMakeAcyclic(Digraph G):
    E' ← ∅
    foreach v ∈ V(G) do
        if |E→(v)| ≥ |E←(v)| then
            E' ← E' ∪ E→(v)

    return G' = (V(G), E')
```



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



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

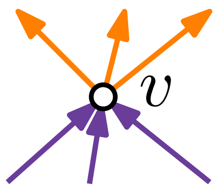
  if $|E^{\rightarrow}(v)| \geq |E^{\leftarrow}(v)|$ **then**

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

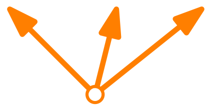
  else

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

return $G' = (V(G), E')$



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



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

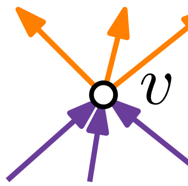
  if $|E^{\rightarrow}(v)| \geq |E^{\leftarrow}(v)|$ **then**

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

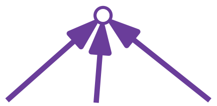
  else

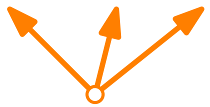
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$

return $G' = (V(G), E')$



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





Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

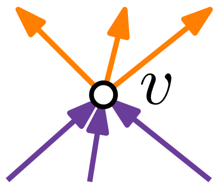
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

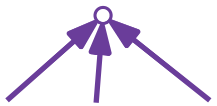
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$

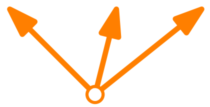
  remove v and $E(v)$ from G.

return $G' = (V(G), E')$



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





Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

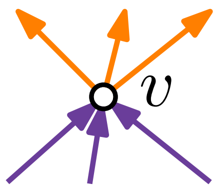
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

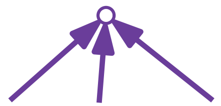
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$

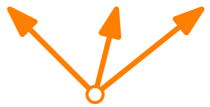
  remove v and $E(v)$ from G.

return $G' = (V(G), E')$



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





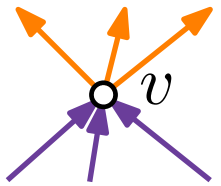
- G' is a DAG.

Heuristic 1

[Berger, Shor '90]

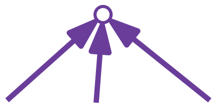
GreedyMakeAcyclic(Digraph G):

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

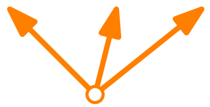


$$
\begin{aligned}
E^{\rightarrow}(v) &:= \{(v,u) \colon (v,u) \in E(G)\} \\
E^{\leftarrow}(v) &:= \{(u,v) \colon (u,v) \in E(G)\} \\
E(v) &:= E^{\rightarrow}(v) \cup E^{\leftarrow}(v)
\end{aligned}
$$

Proof Idea.



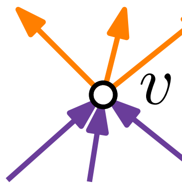
- G' is a DAG.



Heuristic 1

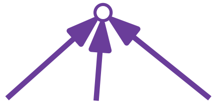
[Berger, Shor '90]

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

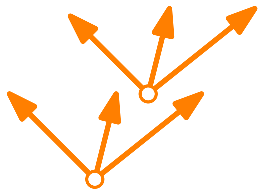


$$E^{\rightarrow}(v) := \{(v,u) : (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v) : (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

Proof Idea.



- G' is a DAG.

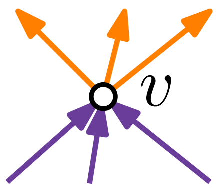


Heuristic 1

[Berger, Shor '90]

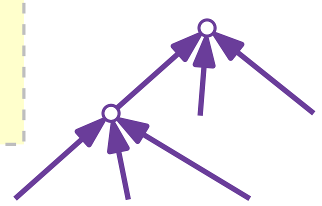
GreedyMakeAcyclic(Digraph G):

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

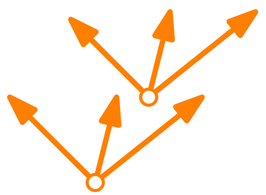


$$
\begin{aligned}
E^{\rightarrow}(v) &:= \{(v,u) \colon (v,u) \in E(G)\} \\
E^{\leftarrow}(v) &:= \{(u,v) \colon (u,v) \in E(G)\} \\
E(v) &:= E^{\rightarrow}(v) \cup E^{\leftarrow}(v)
\end{aligned}
$$

Proof Idea.



- G' is a DAG.



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

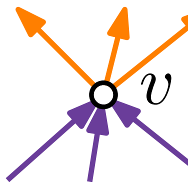
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

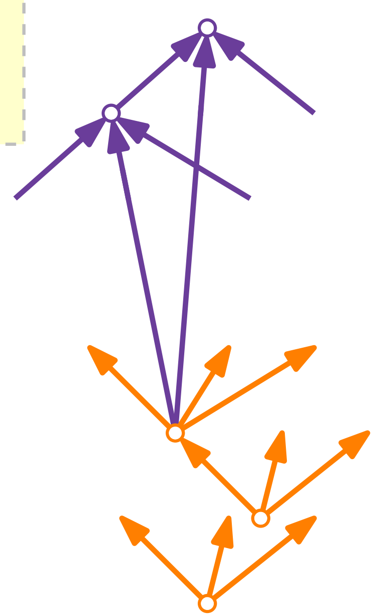
  remove v and $E(v)$ from G.

return $G' = (V(G), E')$



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

Proof Idea.



- G' is a DAG.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

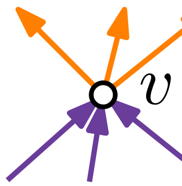
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

return $G' = (V(G), E')$



$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$

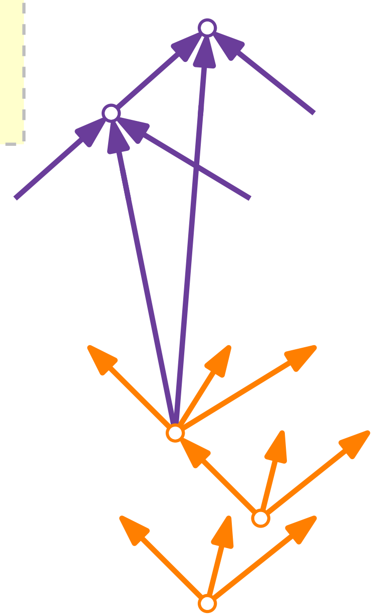
$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$

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

- G' is a DAG.

Proof Idea.

Place the vertices on distinct y-coordinates.



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

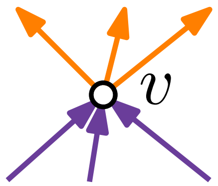
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

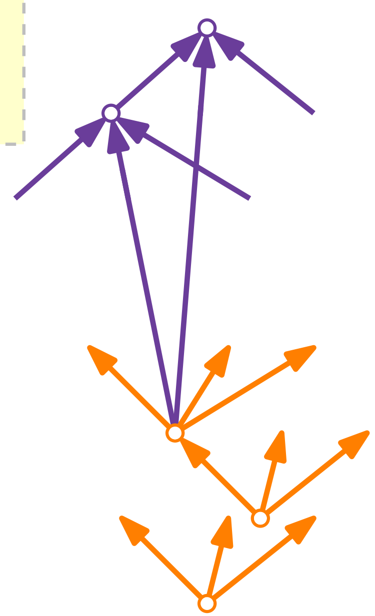
return $G' = (V(G), E')$



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

- G' is a DAG.

Proof Idea.



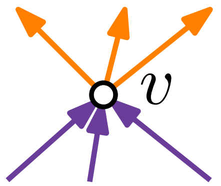
Place the vertices on distinct y-coordinates.

y-coordinates increase/decrease towards the middle.

Heuristic 1

[Berger, Shor '90]

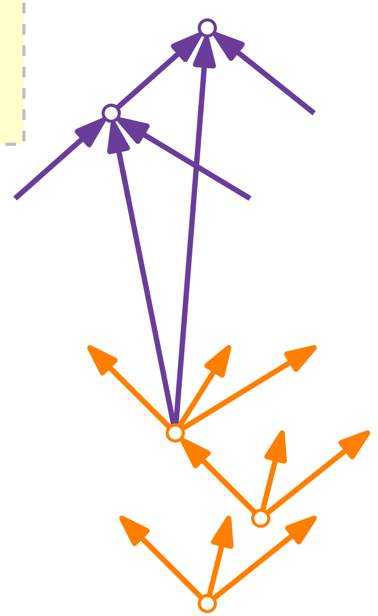
GreedyMakeAcyclic(Digraph G):
  $E' \leftarrow \emptyset$
  foreach $v \in V(G)$ **do**
    if $|E^{\rightarrow}(v)| \geq |E^{\leftarrow}(v)|$ **then**
      $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    else
      $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    remove v and $E(v)$ from G.
  return $G'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



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

- G' is a DAG.

Proof Idea.



Place the vertices on distinct y-coordinates.

y-coordinates increase/decrease towards the middle.

All edges point upwards.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

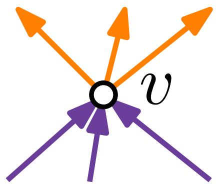
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

return $G' = (V(G), E')$



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

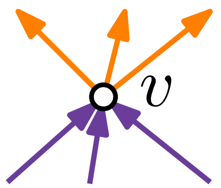
- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

```
E' ← ∅
foreach v ∈ V(G) do
    if |E→(v)| ≥ |E←(v)| then
        E' ← E' ∪ E→(v)
    else
        E' ← E' ∪ E←(v)
    remove v and E(v) from G.
return G' = (V(G), E')
```



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

Proof Idea.

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

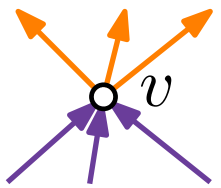
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

return $G' = (V(G), E')$



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

Proof Idea.

Use the vertex order from before (edges in E' upwards).

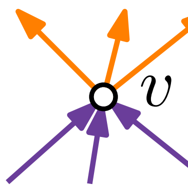
- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

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



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

Proof Idea.

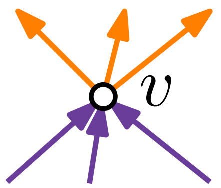
Use the vertex order from before (edges in E' upwards).

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

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

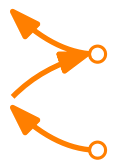


$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

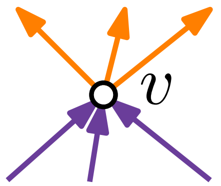
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

return $G' = (V(G), E')$

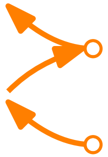


$$
\begin{aligned}
E^{\rightarrow}(v) &:= \{(v,u) \colon (v,u) \in E(G)\} \\
E^{\leftarrow}(v) &:= \{(u,v) \colon (u,v) \in E(G)\} \\
E(v) &:= E^{\rightarrow}(v) \cup E^{\leftarrow}(v)
\end{aligned}
$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

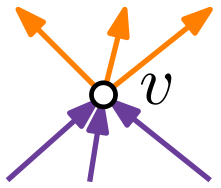
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

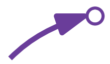
return $G' = (V(G), E')$



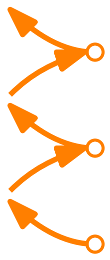
$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).



- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

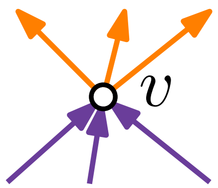
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

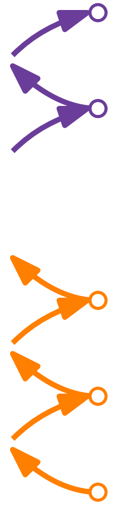
return $G' = (V(G), E')$



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

Proof Idea.

Use the vertex order from before (edges in E' upwards).



- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

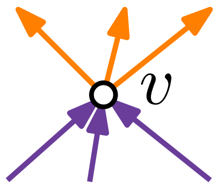
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

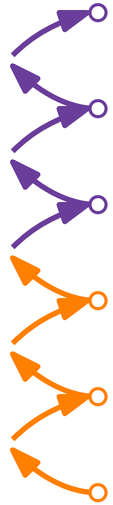
return $G' = (V(G), E')$



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

Proof Idea.

Use the vertex order from before (edges in E' upwards).

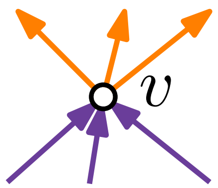


- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

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

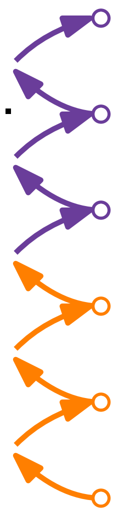


$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

In this order, add the edges of $E(G) \setminus E'$ in rev. direction.



- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

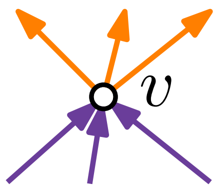
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

return $G' = (V(G), E')$

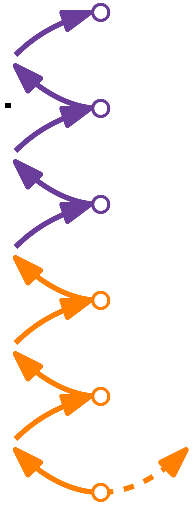


$$
\begin{aligned}
E^{\rightarrow}(v) &:= \{(v,u) \colon (v,u) \in E(G)\} \\
E^{\leftarrow}(v) &:= \{(u,v) \colon (u,v) \in E(G)\} \\
E(v) &:= E^{\rightarrow}(v) \cup E^{\leftarrow}(v)
\end{aligned}
$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

In this order, add the edges of $E(G) \setminus E'$ in rev. direction.

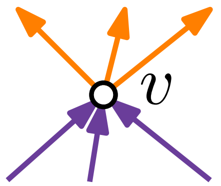


- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

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

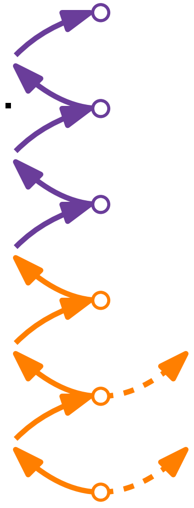


$$
\begin{aligned}
E^{\rightarrow}(v) &:= \{(v,u) \colon (v,u) \in E(G)\} \\
E^{\leftarrow}(v) &:= \{(u,v) \colon (u,v) \in E(G)\} \\
E(v) &:= E^{\rightarrow}(v) \cup E^{\leftarrow}(v)
\end{aligned}
$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

In this order, add the edges of $E(G) \setminus E'$ in rev. direction.



- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

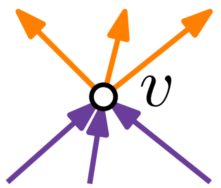
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

return $G' = (V(G), E')$

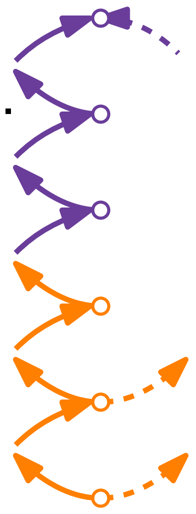


$$E^{\rightarrow}(v) := \{(v,u)\colon (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v)\colon (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

In this order, add the edges of $E(G) \setminus E'$ in rev. direction.



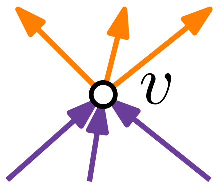
- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

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

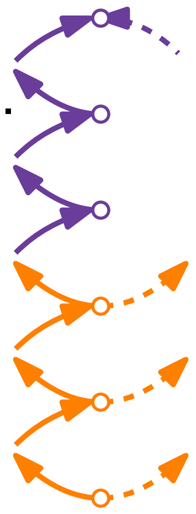


$$
\begin{aligned}
E^{\rightarrow}(v) &:= \{(v,u) \colon (v,u) \in E(G)\} \\
E^{\leftarrow}(v) &:= \{(u,v) \colon (u,v) \in E(G)\} \\
E(v) &:= E^{\rightarrow}(v) \cup E^{\leftarrow}(v)
\end{aligned}
$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

In this order, add the edges of $E(G) \setminus E'$ in rev. direction.

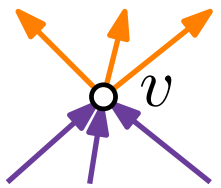


- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

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

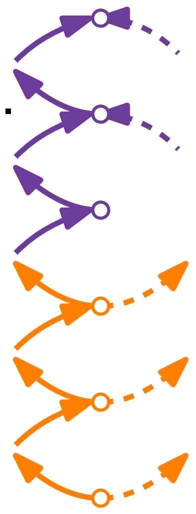


$$E^{\rightarrow}(v) := \{(v,u) : (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v) : (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

Proof Idea.

Use the vertex order from before (edges in E' upwards).

In this order, add the edges of $E(G) \setminus E'$ in rev. direction.



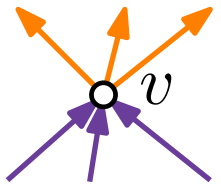
- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

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



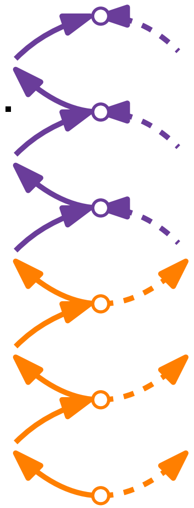
$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Proof Idea.

Use the vertex order from before (edges in E' upwards).

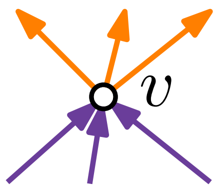
In this order, add the edges of $E(G) \setminus E'$ in rev. direction.



Heuristic 1

[Berger, Shor '90]

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



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

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

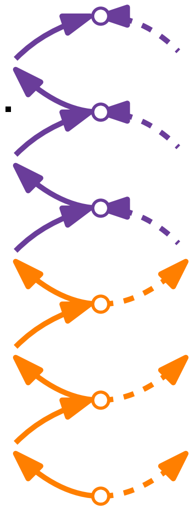
Proof Idea.

Use the vertex order from before (edges in E' upwards).

In this order, add the edges of $E(G) \setminus E'$ in rev. direction.

Added edges have other endpoint more in the middle.

$\rightarrow$ All edges point upwards.



Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

$E' \leftarrow \emptyset$

foreach $v \in V(G)$ **do**

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

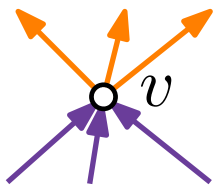
  $E' \leftarrow E' \cup E^{\rightarrow}(v)$

 else

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

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

return $G' = (V(G), E')$



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

- Runtime:

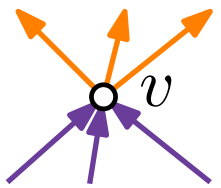
- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

```
E' ← ∅
foreach v ∈ V(G) do
    if |E→(v)| ≥ |E←(v)| then
        E' ← E' ∪ E→(v)
    else
        E' ← E' ∪ E←(v)
    remove v and E(v) from G.
return G' = (V(G), E')
```



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

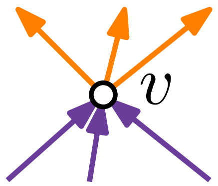
- Runtime: $\mathcal{O}(|V(G)| + |E(G)|)$

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 1

[Berger, Shor '90]

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



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

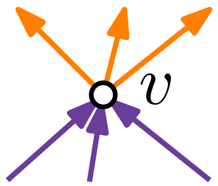
- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.
- Runtime: $\mathcal{O}(|V(G)| + |E(G)|)$
- Quality guarantee: $|E'| \geq$

Heuristic 1

[Berger, Shor '90]

GreedyMakeAcyclic(Digraph G):

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



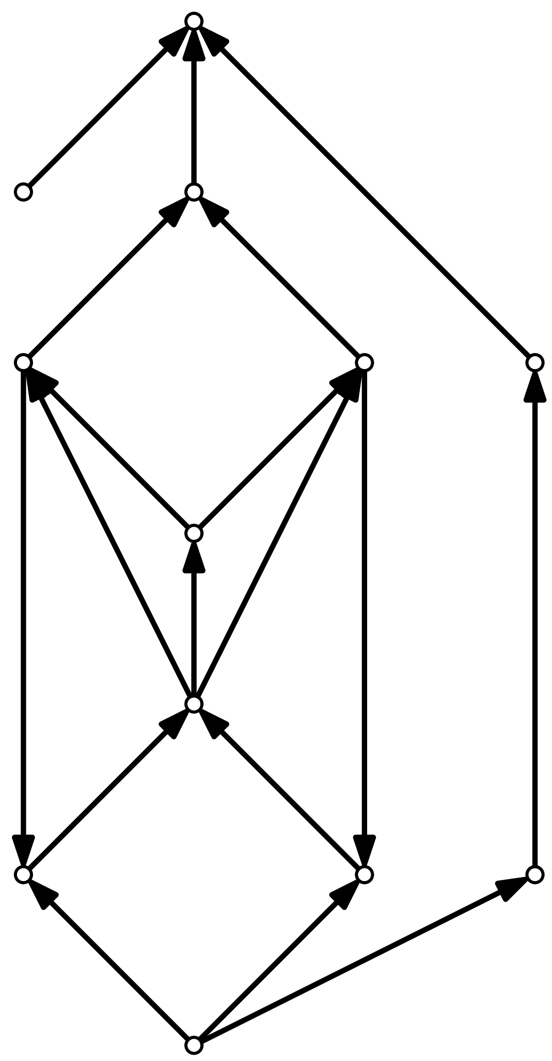
$$E^{\rightarrow}(v) := \{(v,u) \colon (v,u) \in E(G)\}$$
$$E^{\leftarrow}(v) := \{(u,v) \colon (u,v) \in E(G)\}$$
$$E(v) := E^{\rightarrow}(v) \cup E^{\leftarrow}(v)$$

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

- G' is a DAG.
- $E(G) \setminus E'$ is a feedback set.

Heuristic 2

[Eades, Lin, Smyth '93]



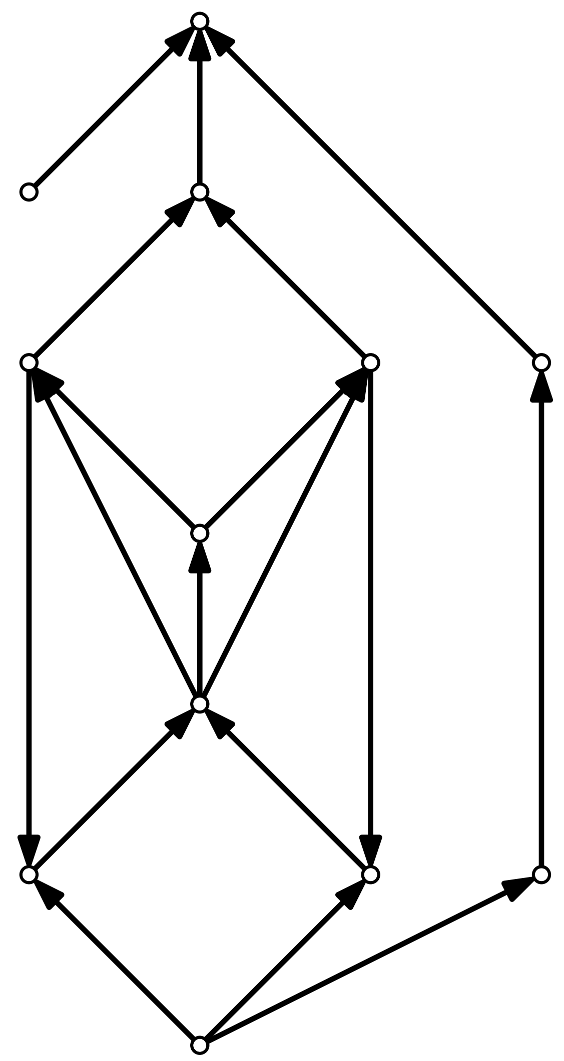
Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

return $G' = (V(G), E')$



Heuristic 2

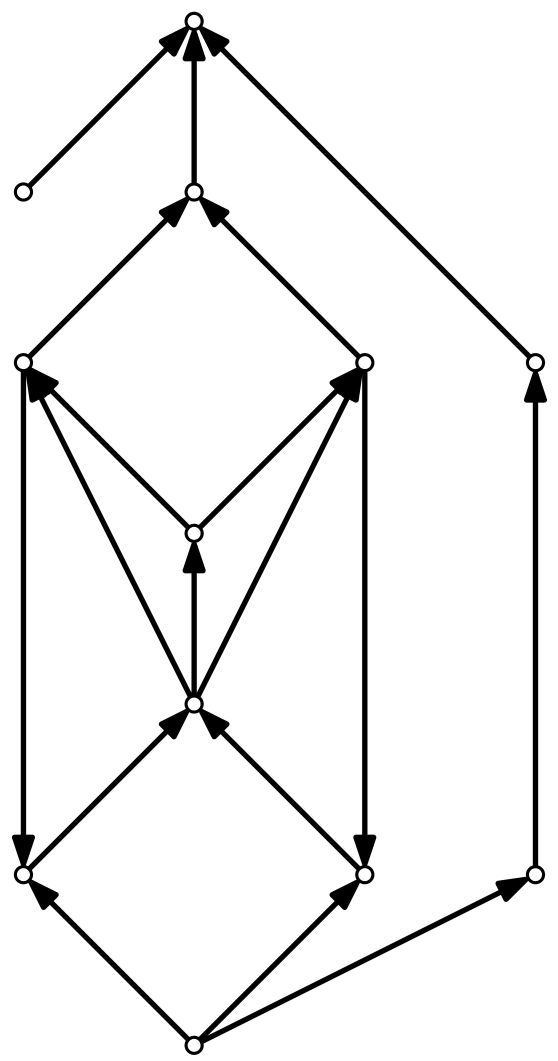
[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

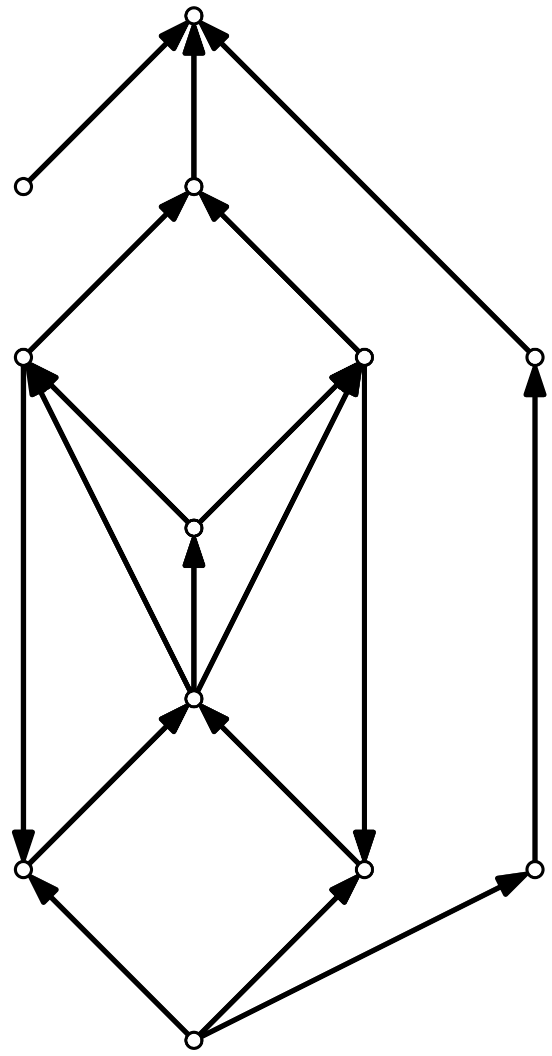
GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

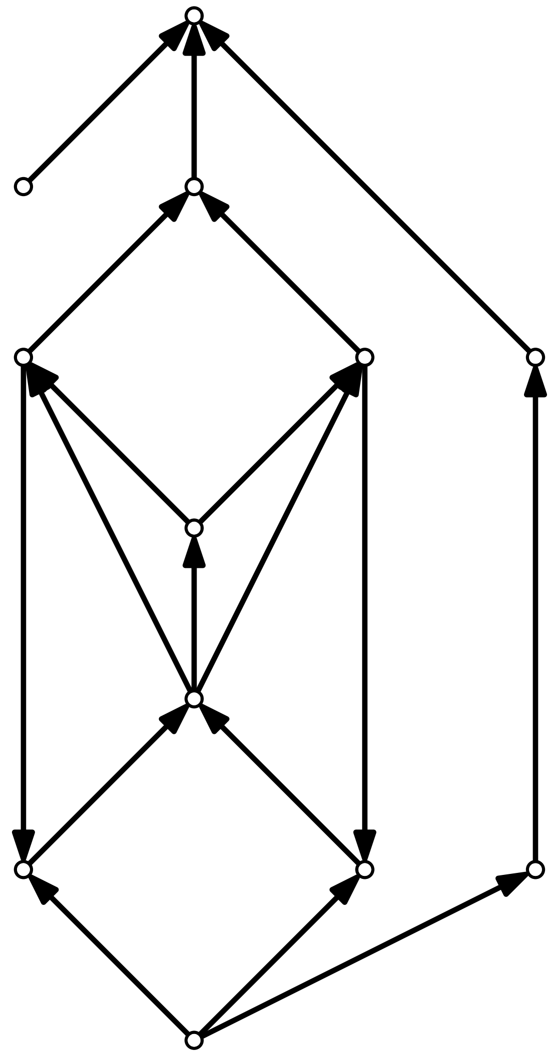
while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

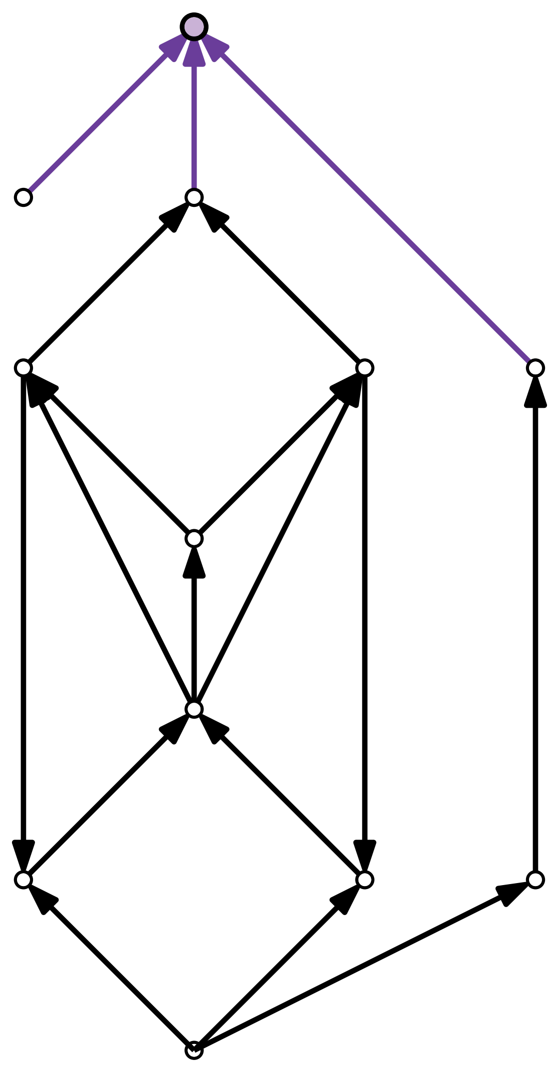
while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

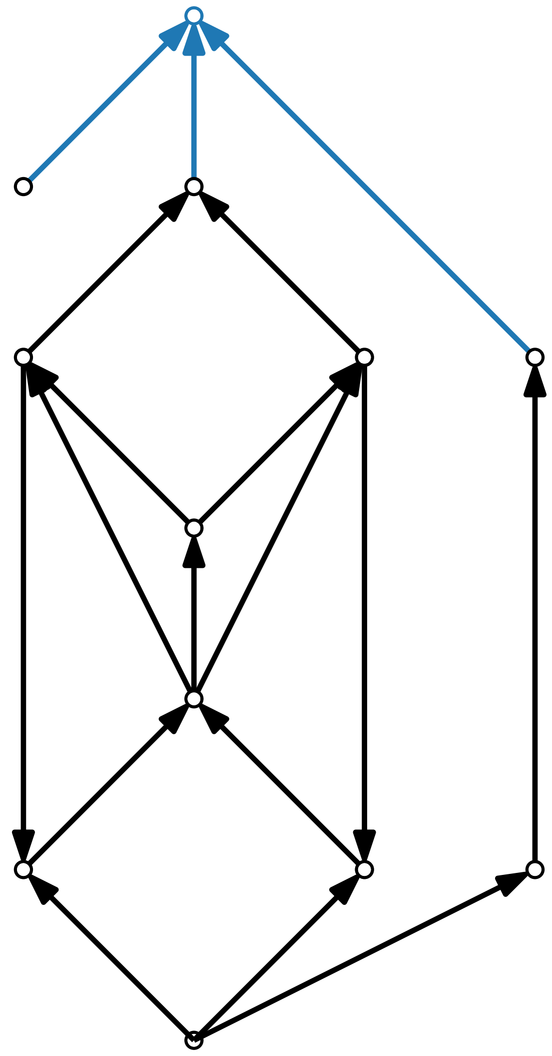
while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



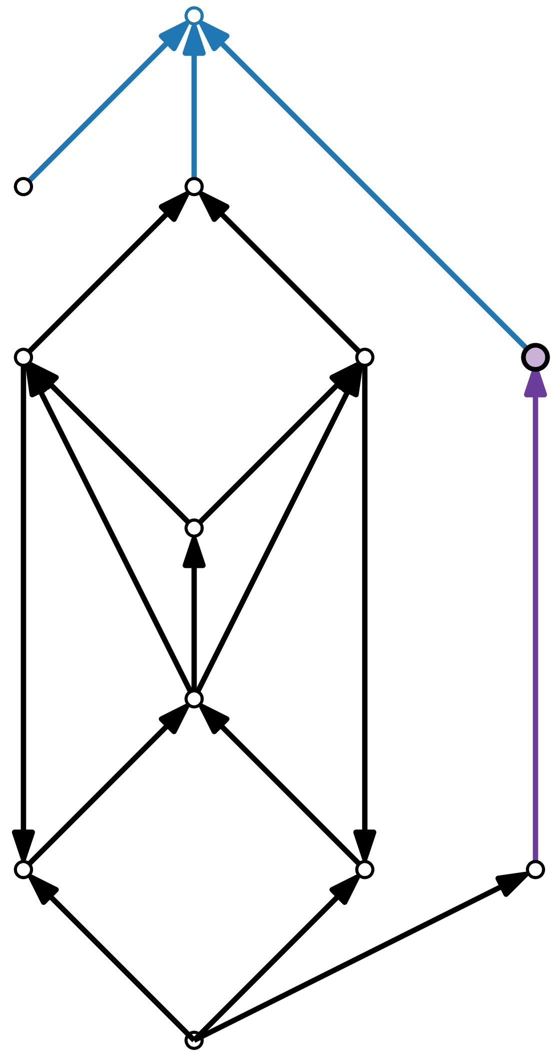
Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



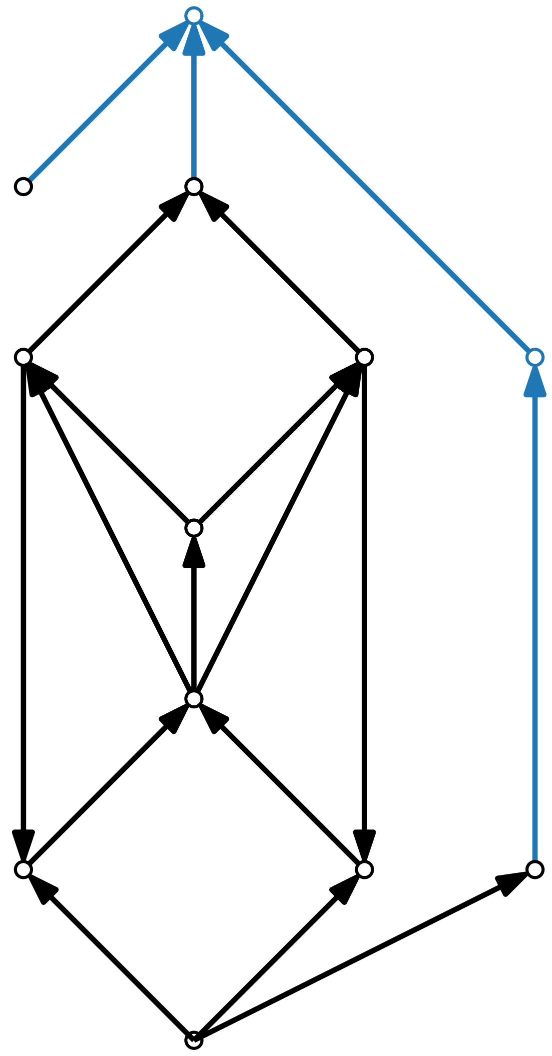
Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

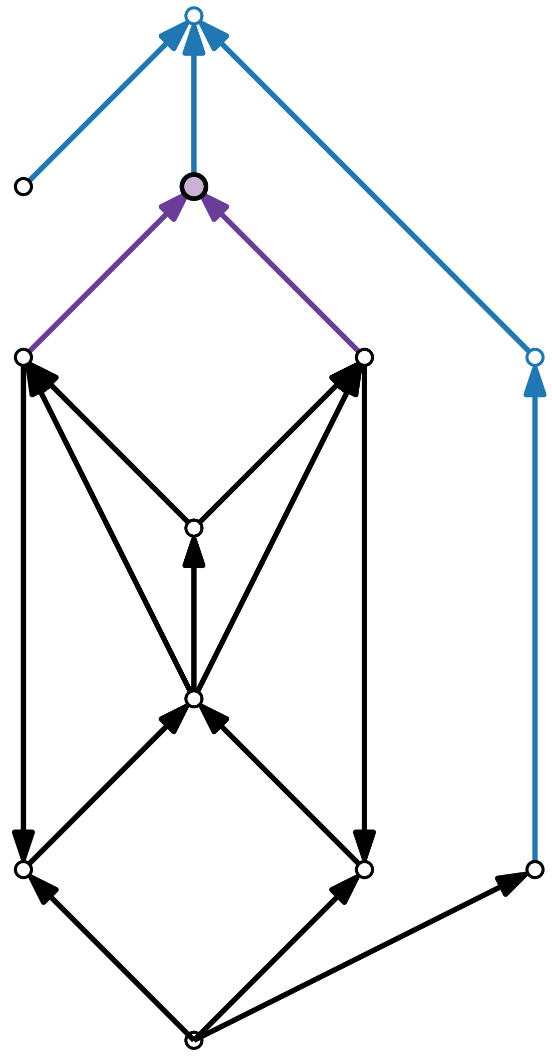
while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

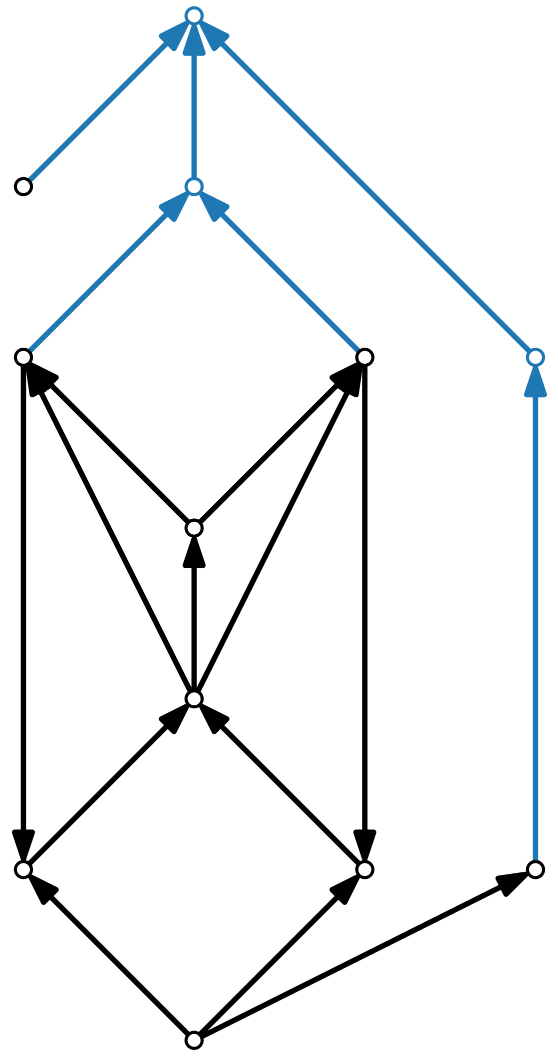
while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

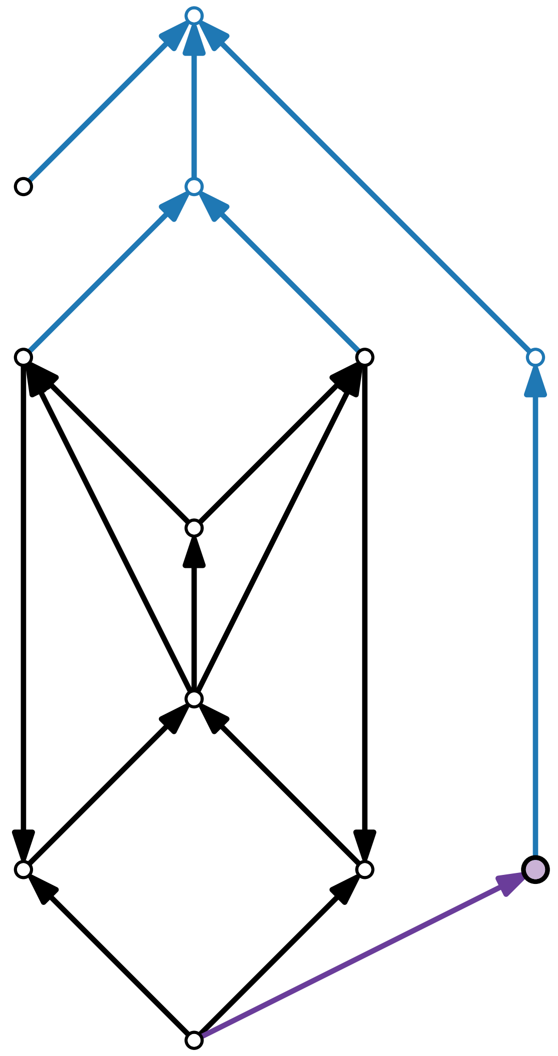
while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



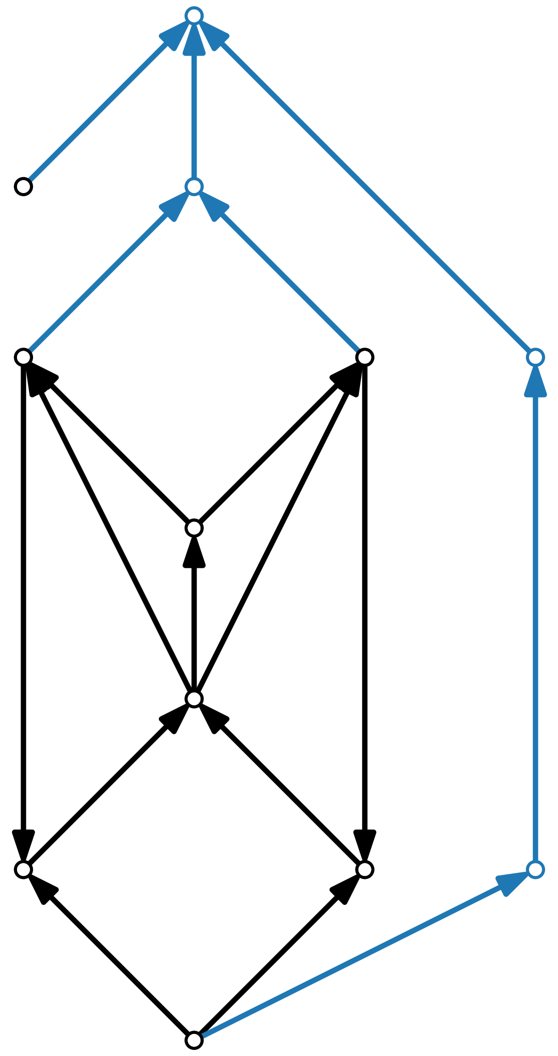
Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

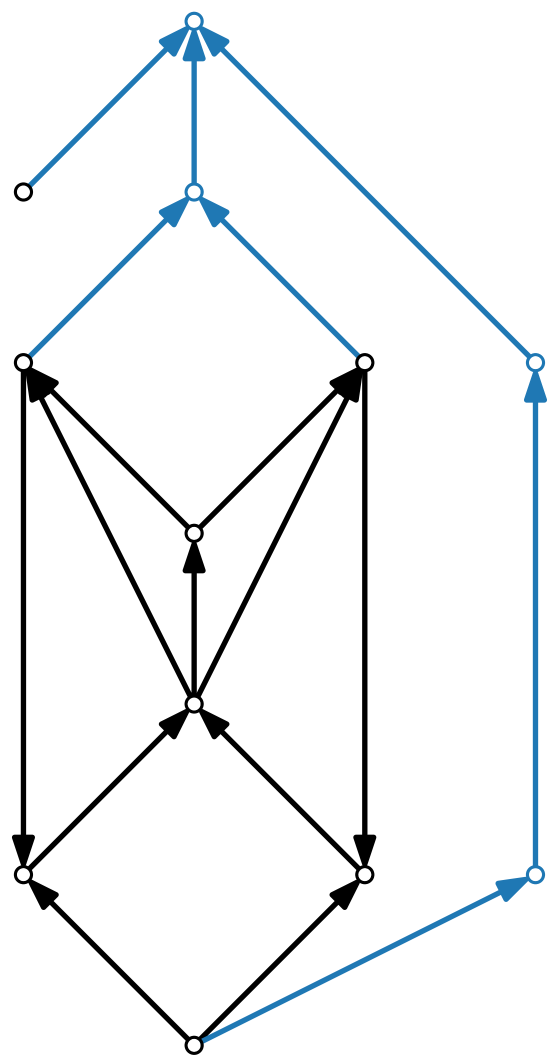
  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

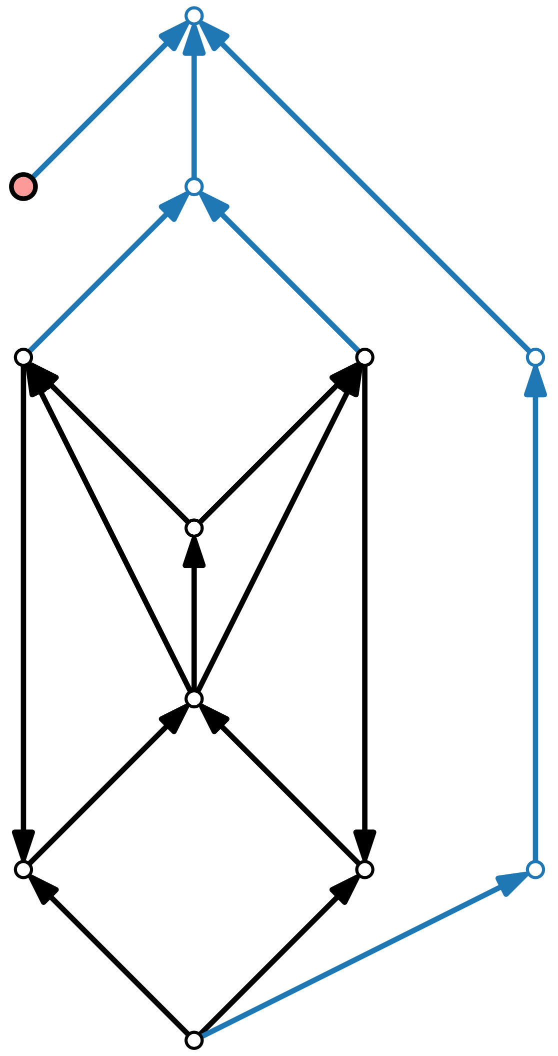
  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

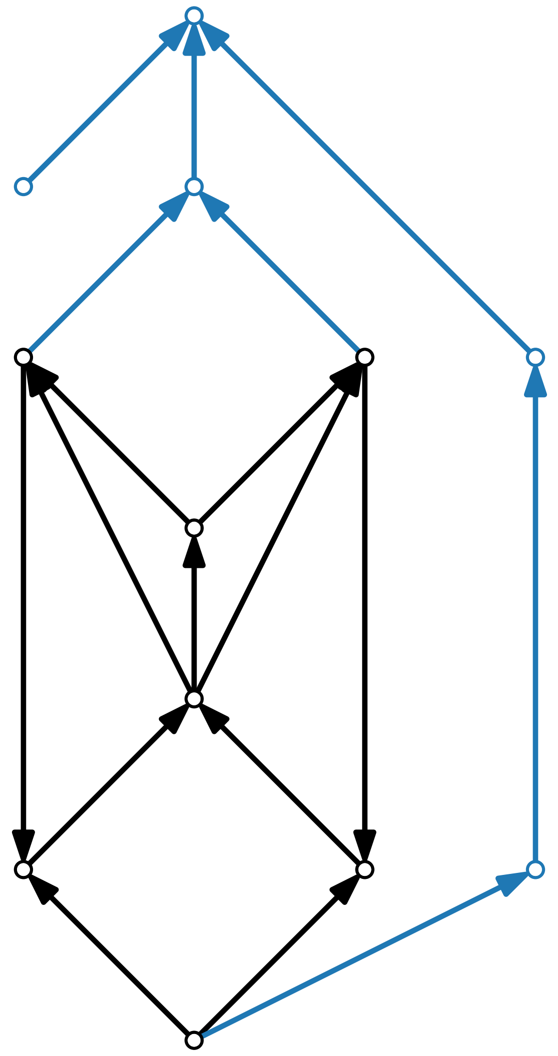
  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

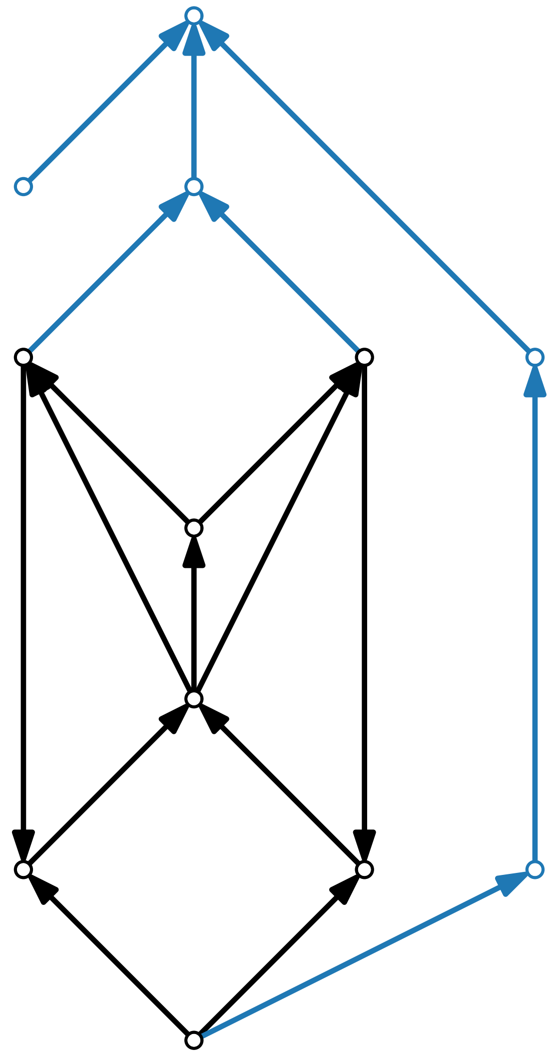
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

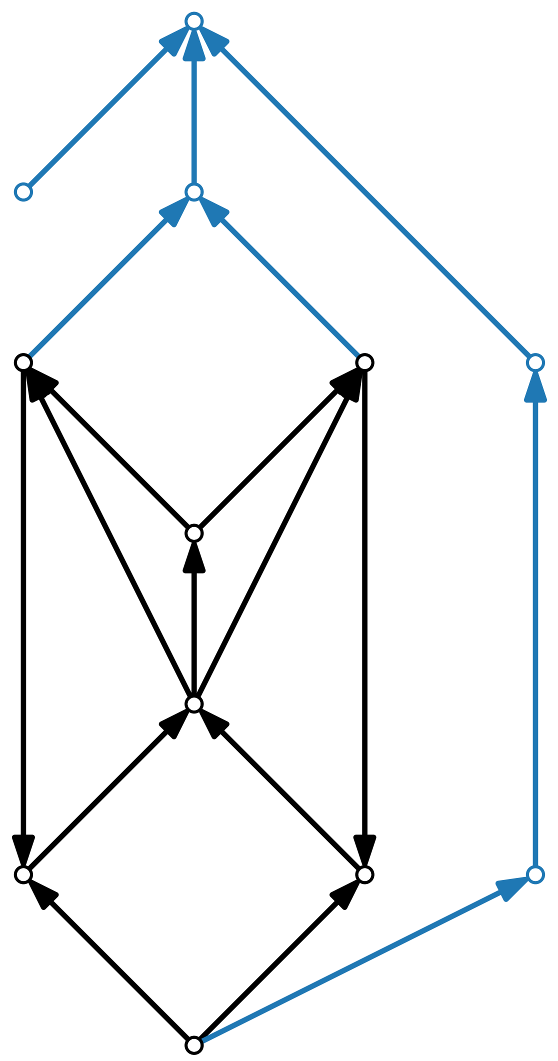
  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

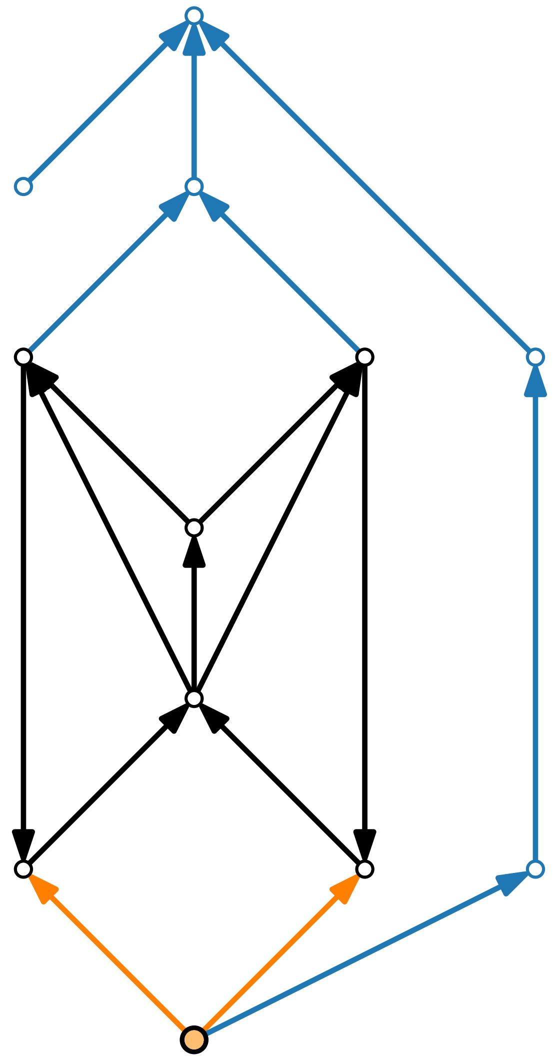
  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

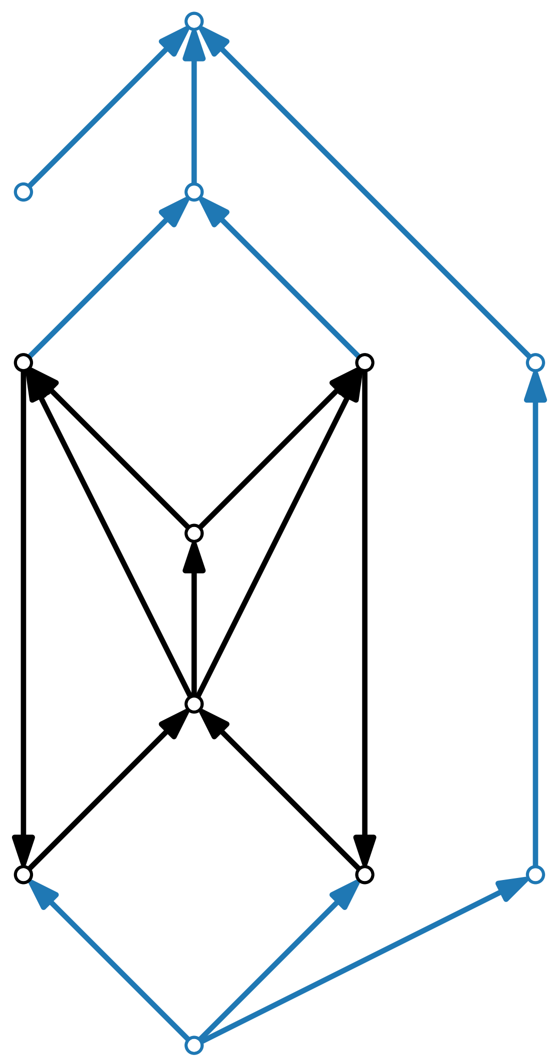
  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

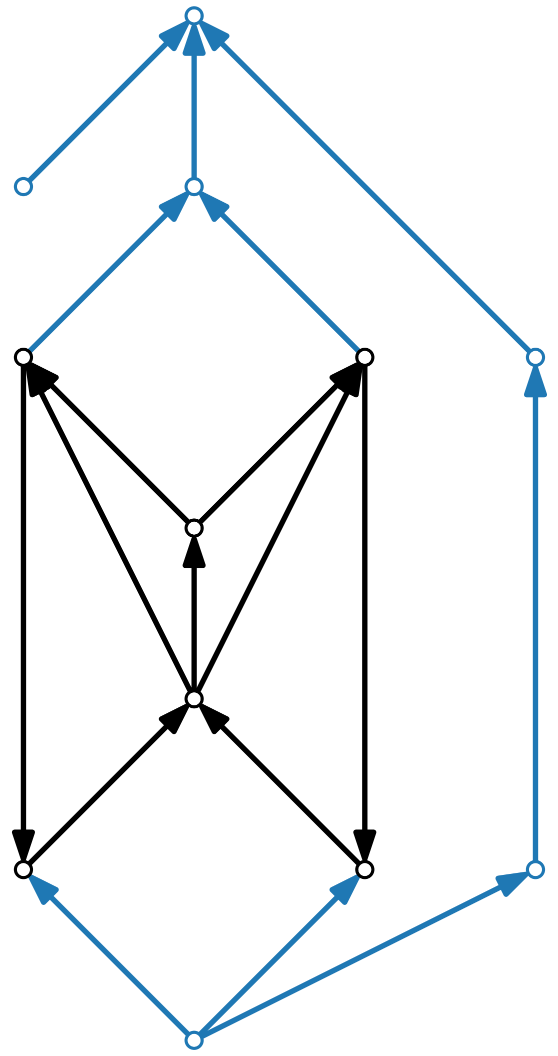
  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

  if $V(G) \neq \emptyset$ **then**

return $G' = (V(G), E')$

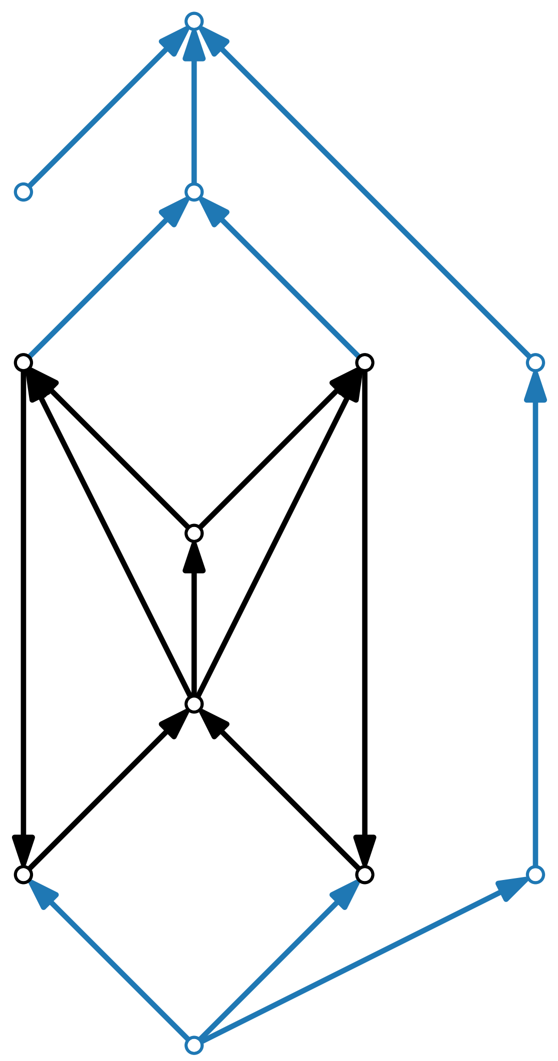


Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.
  Remove all isolated vertices from $V(G)$.
  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.
  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

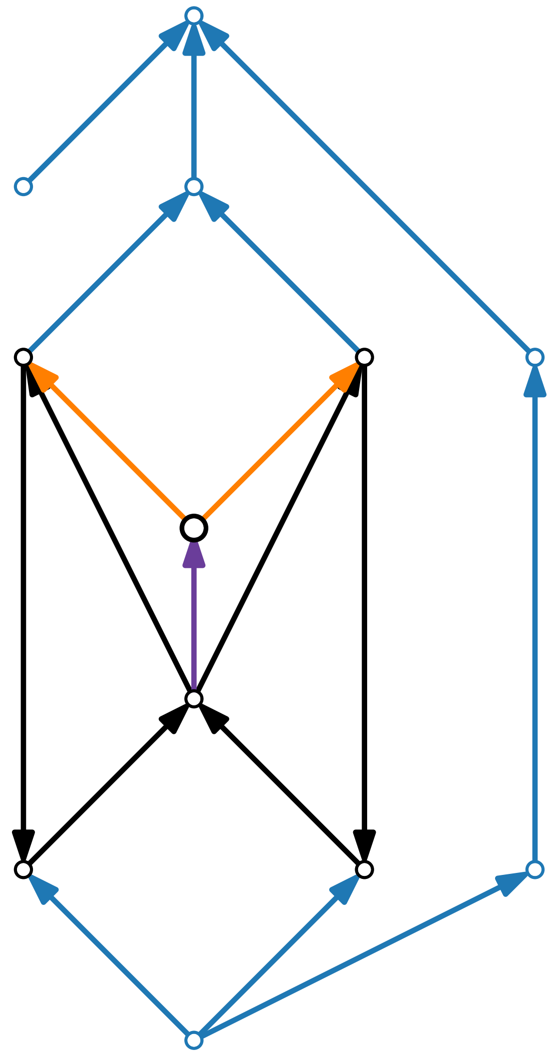
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

    Remove v and $E^{\rightarrow}(v)$.

  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

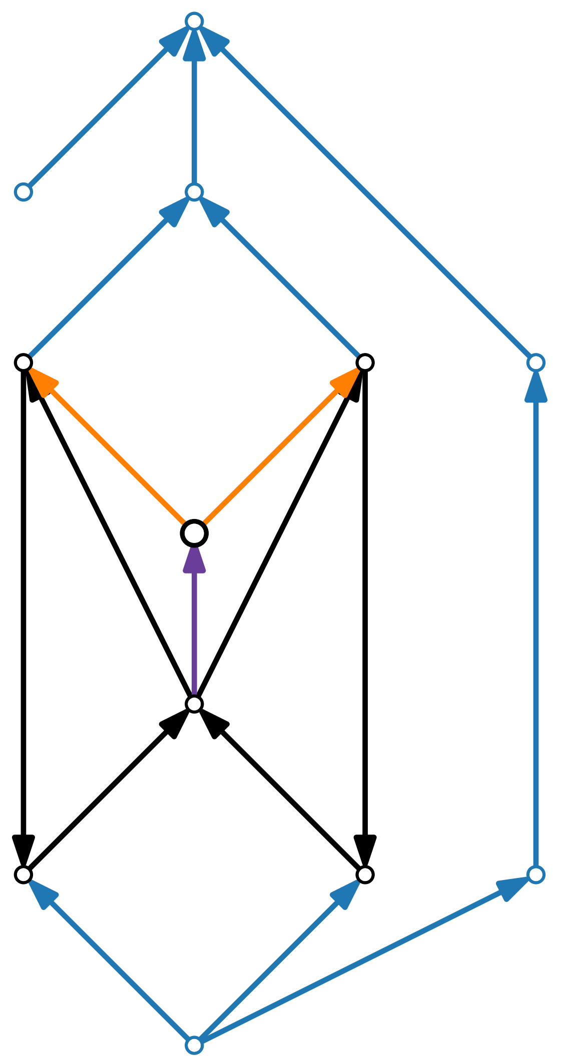
    Remove v and $E^{\rightarrow}(v)$.

  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

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

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

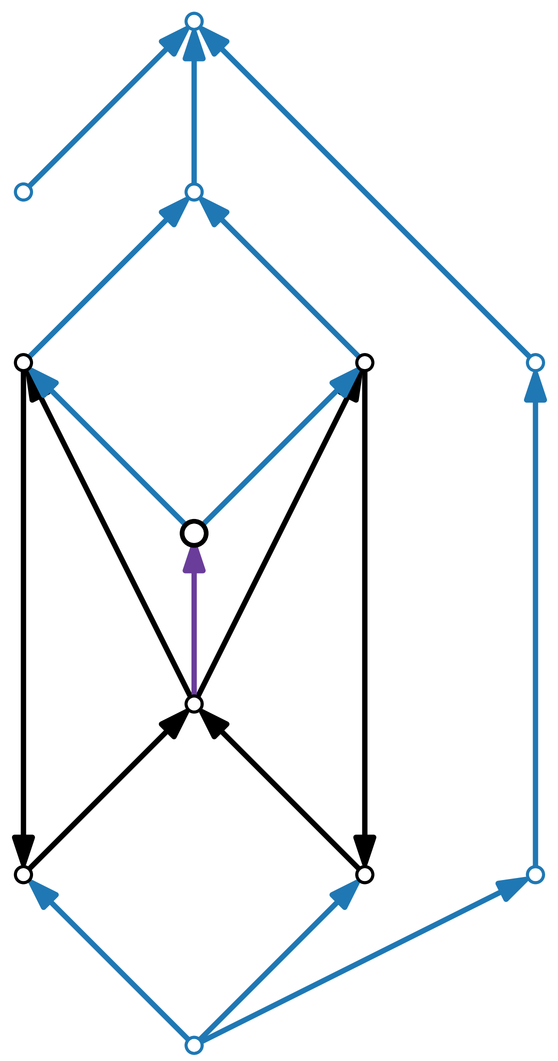
```
GreedyMakeAcyclic2(Digraph G):
  E' ← ∅
  while V(G) ≠ ∅ do
      while V(G) contains a sink v do
          E' ← E' ∪ E←(v)
          Remove v and E←(v).

      Remove all isolated vertices from V(G).

      while V(G) contains a source v do
          E' ← E' ∪ E→(v)
          Remove v and E→(v).

      if V(G) ≠ ∅ then
          Let v ∈ V(G) such that |E→(v)| − |E←(v)| maximal.
          E' ← E' ∪ E→(v)

  return G' = (V(G), E')
```



Heuristic 2

[Eades, Lin, Smyth '93]

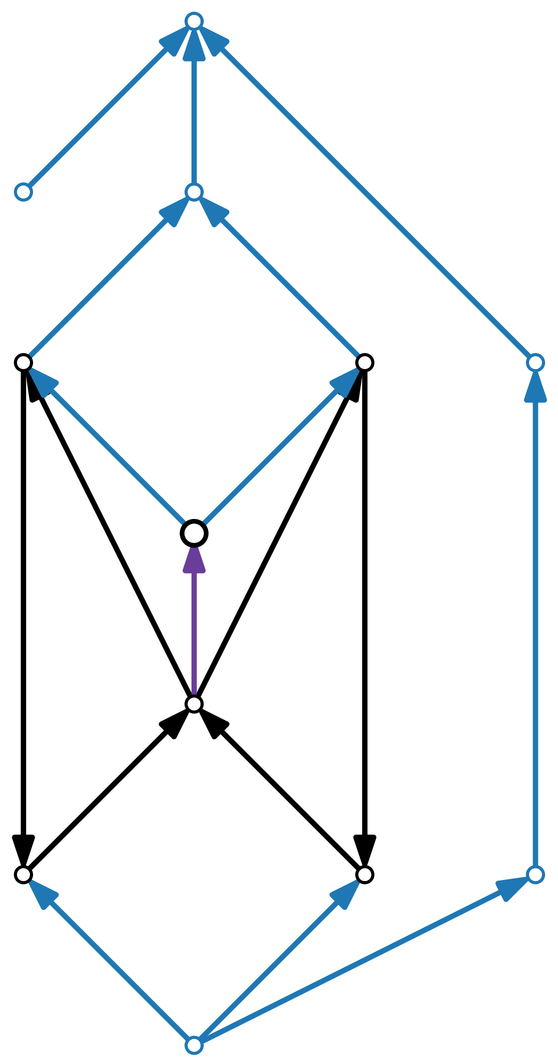
GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.

  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.
return $G' = (V(G), E')$

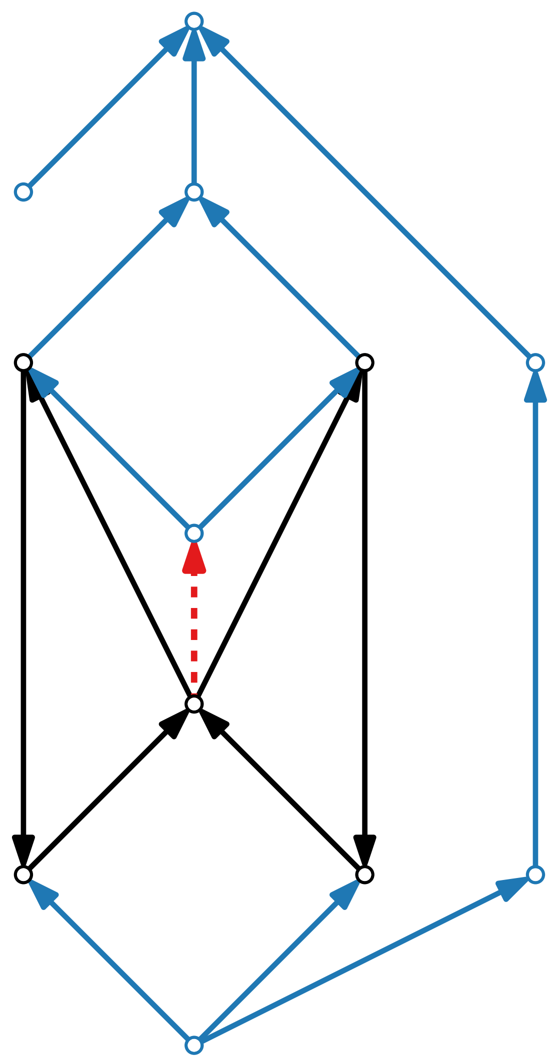


Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.
  Remove all isolated vertices from $V(G)$.
  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.
  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.
return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

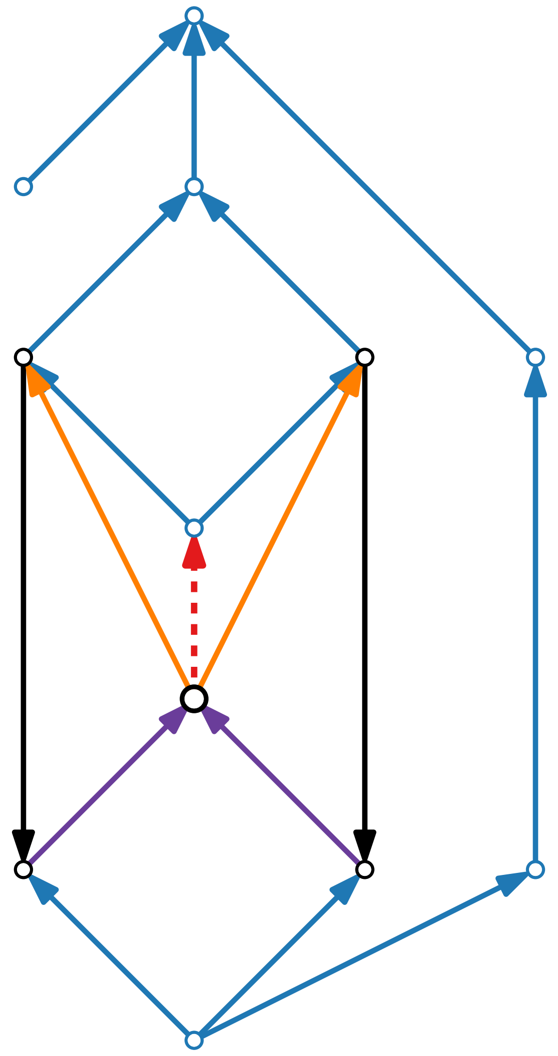
  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

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

    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$

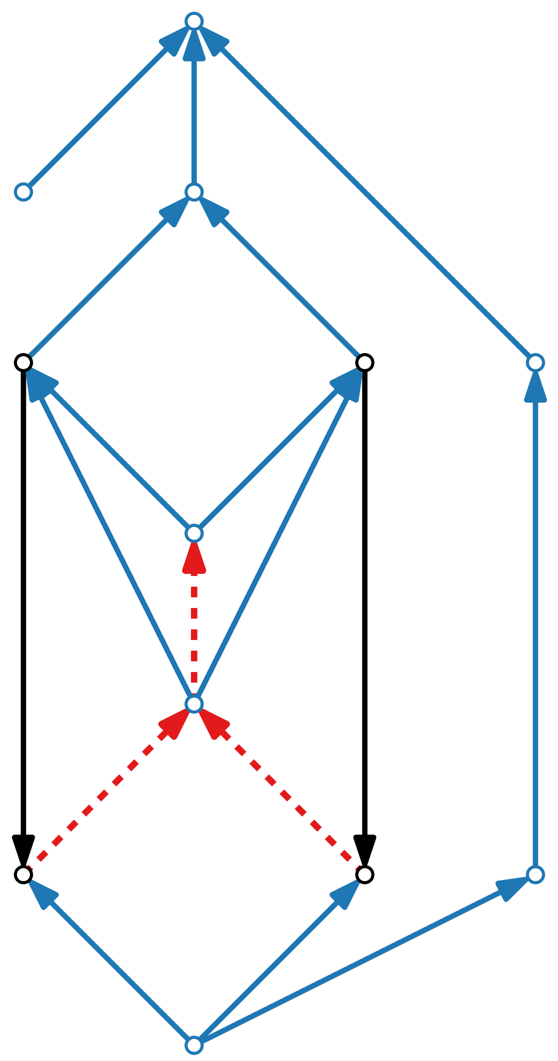


Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.
  Remove all isolated vertices from $V(G)$.
  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.
  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.
return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

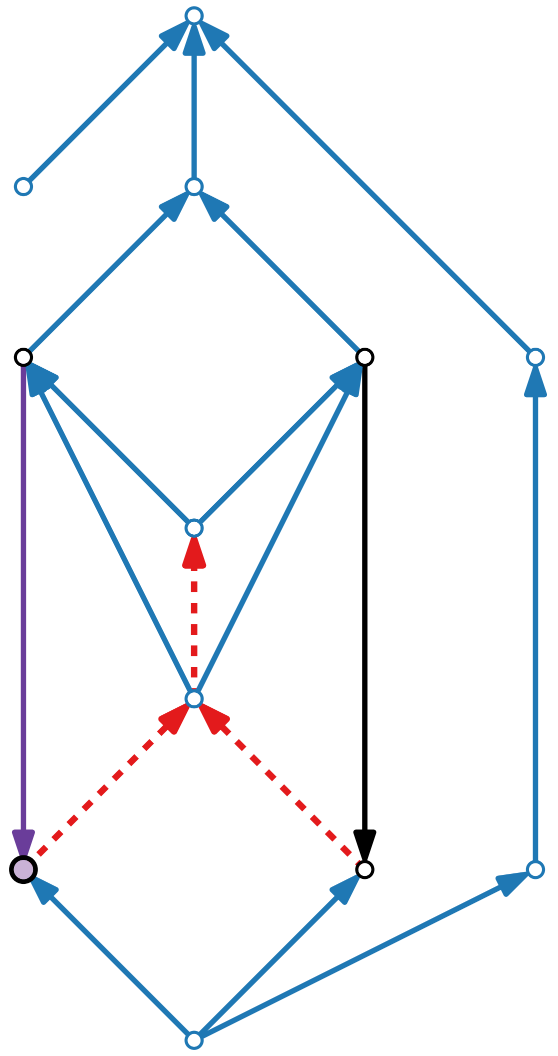
GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.

  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.
return $G' = (V(G), E')$

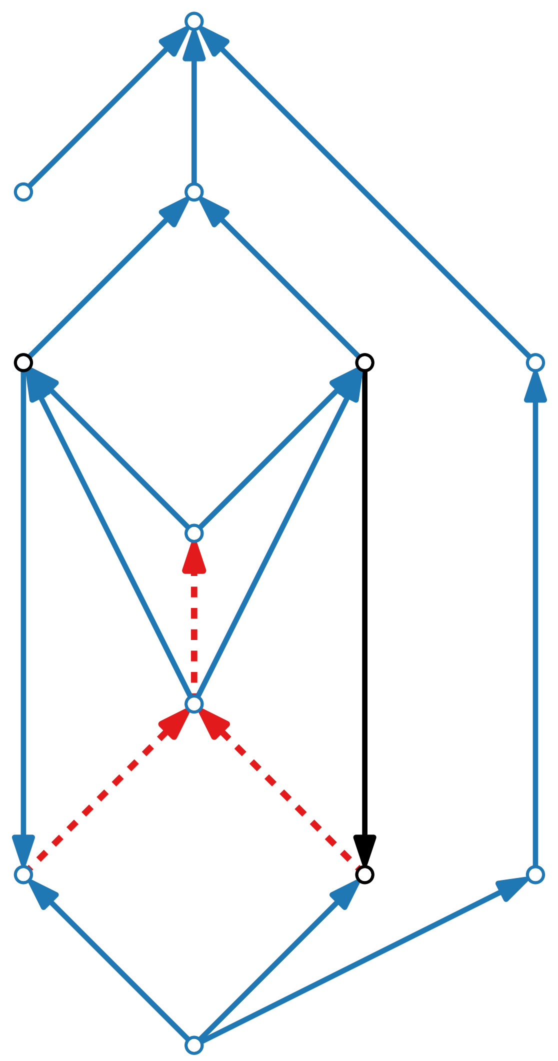


Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.
  Remove all isolated vertices from $V(G)$.
  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.
  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.
return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

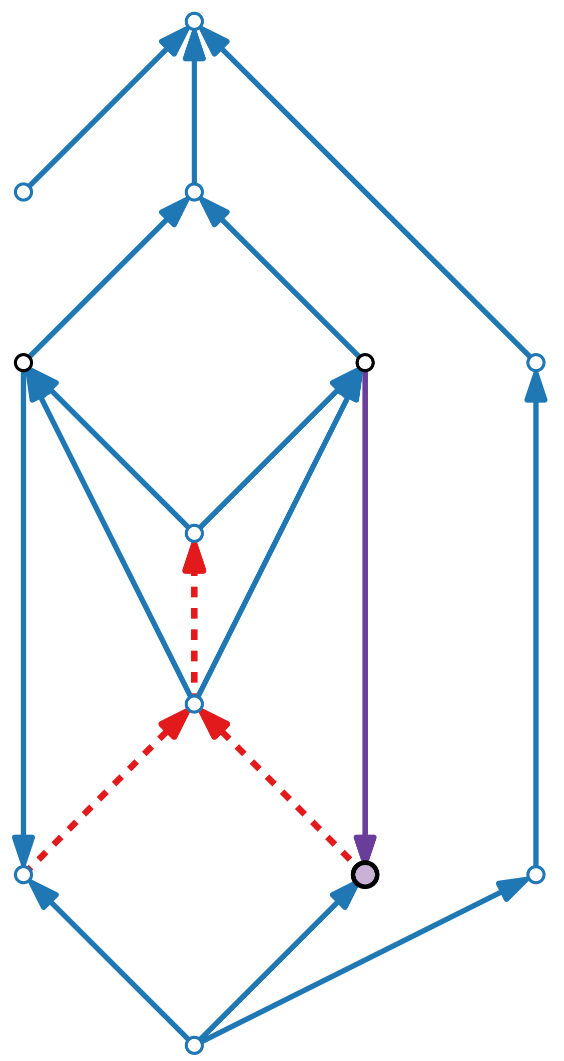
  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

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

    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

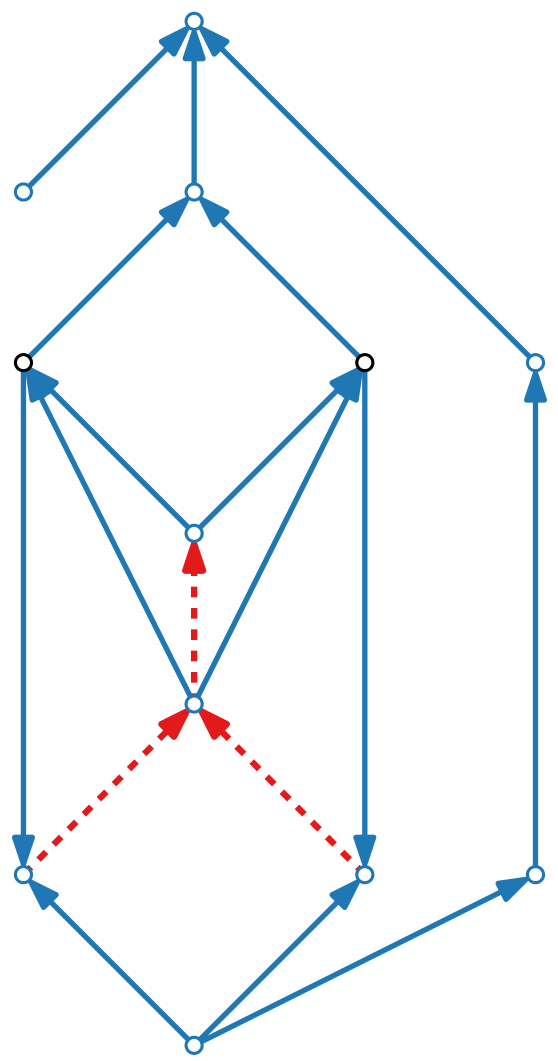
  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

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

    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

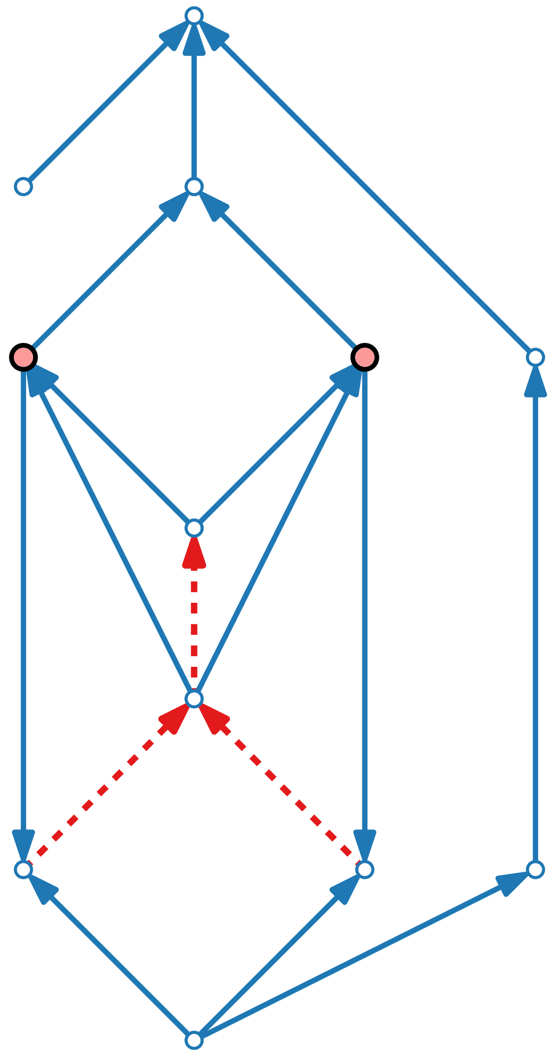
$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.

  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

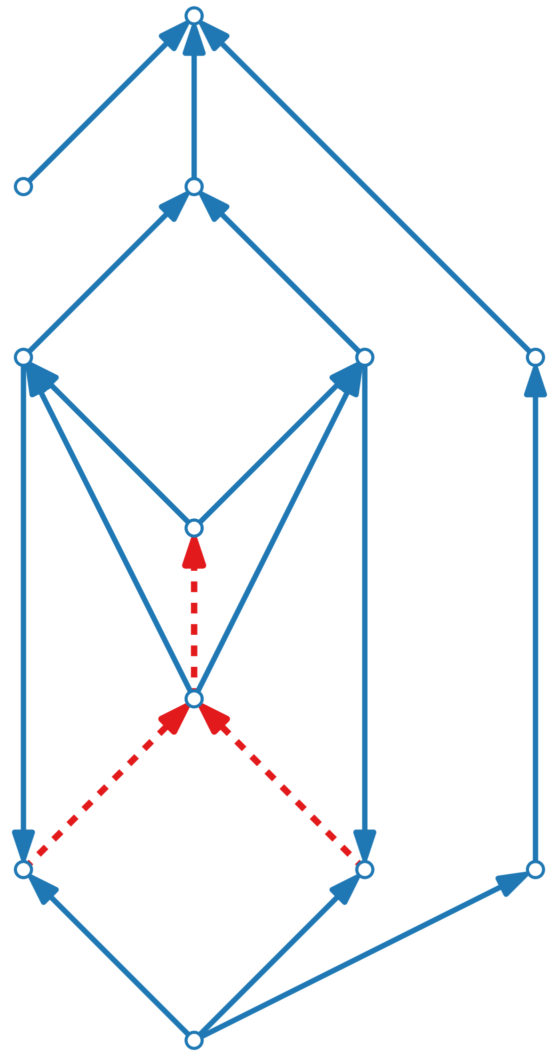
$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.

  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

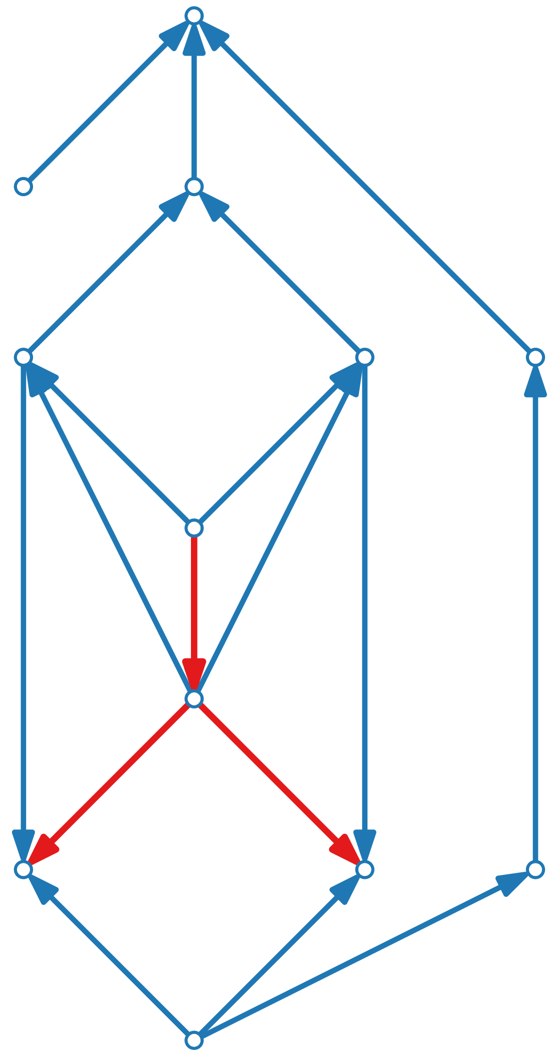
  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

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

    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$



Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

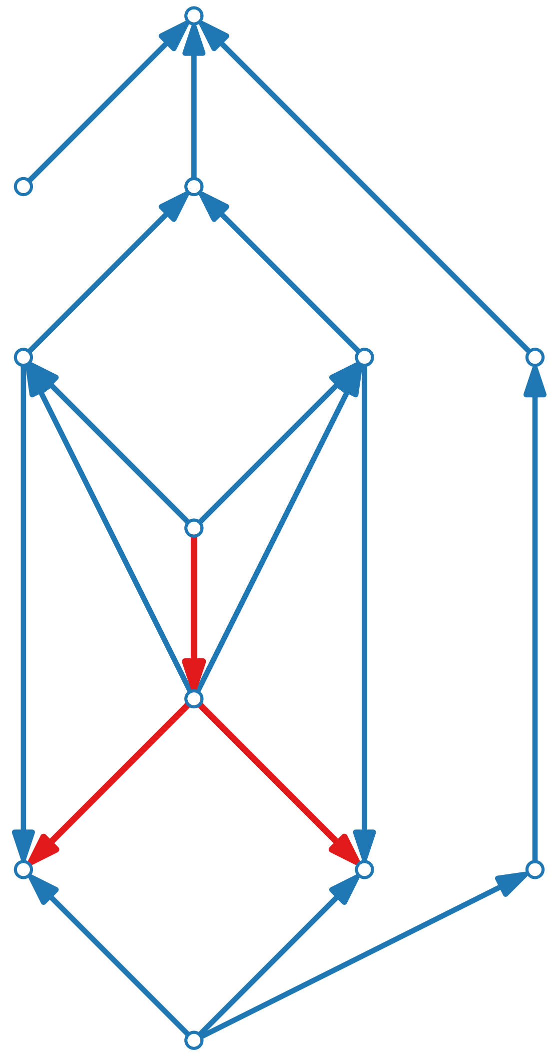
  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

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

    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$



- Time:

Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$

while $V(G) \neq \emptyset$ **do**

  while $V(G)$ contains a sink v **do**

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

    Remove v and $E^{\leftarrow}(v)$.

  Remove all isolated vertices from $V(G)$.

  while $V(G)$ contains a source v **do**

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

    Remove v and $E^{\rightarrow}(v)$.

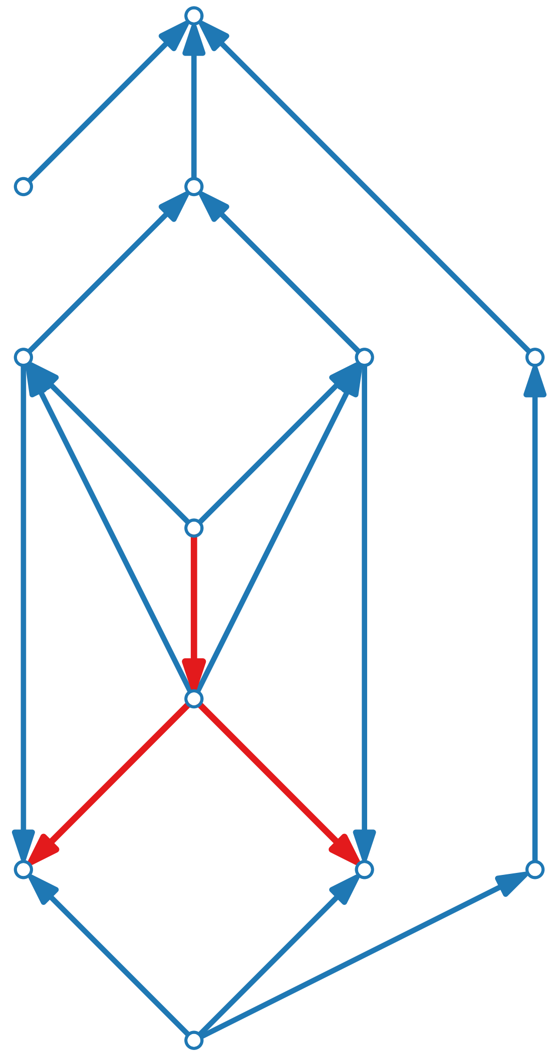
  if $V(G) \neq \emptyset$ **then**

    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.

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

    Remove v and $E(v)$ from G.

return $G' = (V(G), E')$



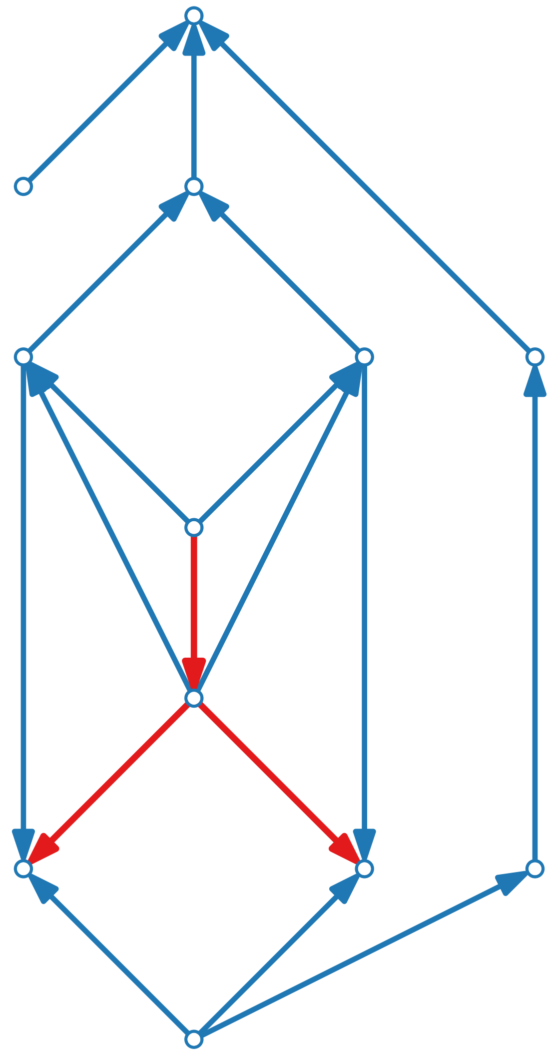
- Time: $\mathcal{O}(|V(G)| + |E(G)|)$

Heuristic 2

[Eades, Lin, Smyth '93]

GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.
  Remove all isolated vertices from $V(G)$.
  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.
  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.
return $G' = (V(G), E')$



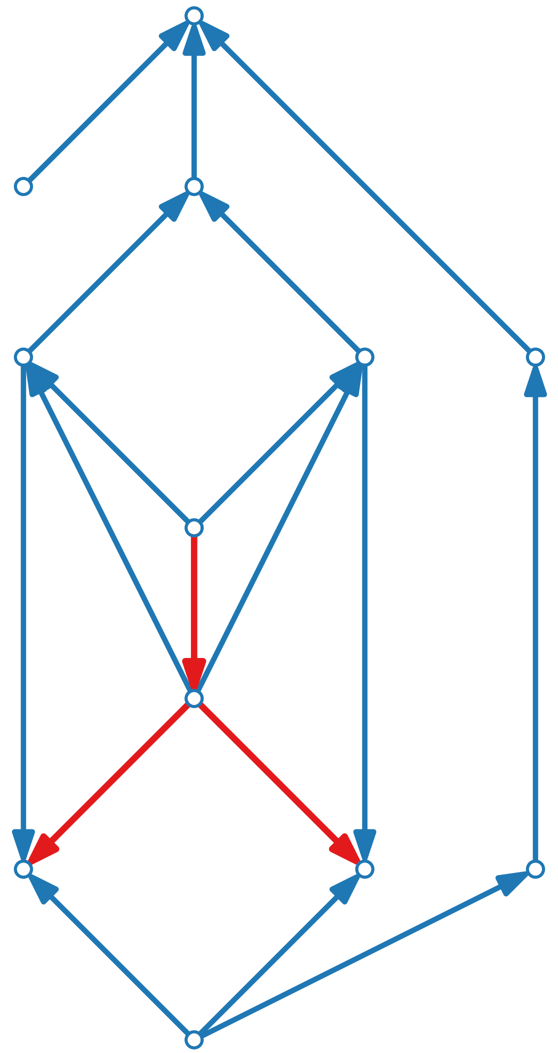
- Time: $\mathcal{O}(|V(G)| + |E(G)|)$ [Main idea: Use bins for sinks and sources, and a bin for each $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$]

Heuristic 2

[Eades, Lin, Smyth '93]

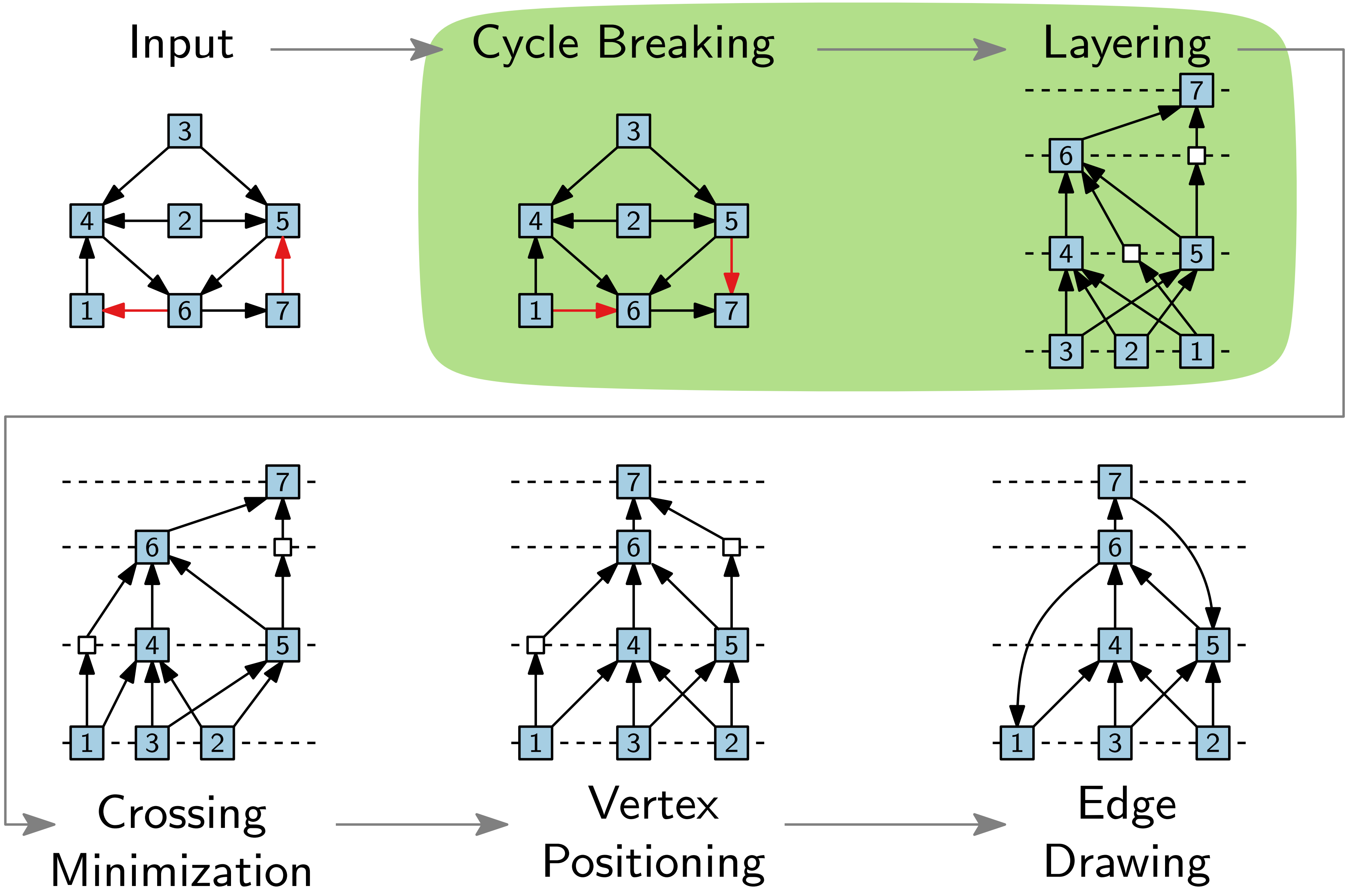
GreedyMakeAcyclic2(Digraph G):

$E' \leftarrow \emptyset$
while $V(G) \neq \emptyset$ **do**
  while $V(G)$ contains a sink v **do**
    $E' \leftarrow E' \cup E^{\leftarrow}(v)$
    Remove v and $E^{\leftarrow}(v)$.
  Remove all isolated vertices from $V(G)$.
  while $V(G)$ contains a source v **do**
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E^{\rightarrow}(v)$.
  if $V(G) \neq \emptyset$ **then**
    Let $v \in V(G)$ such that $|E^{\rightarrow}(v)| - |E^{\leftarrow}(v)|$ maximal.
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$
    Remove v and $E(v)$ from G.
return $G' = (V(G), E')$

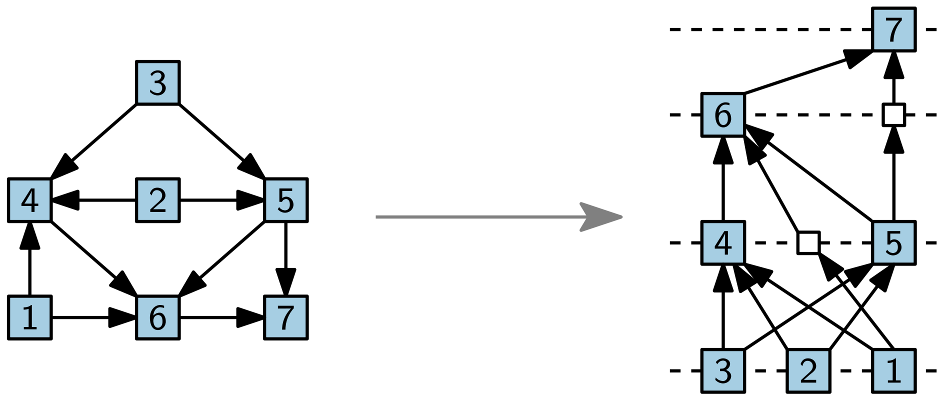


- Time: $\mathcal{O}(|V(G)| + |E(G)|)$ [Main idea: Use bins for sinks and sources, and a bin for each $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$]
- Quality guarantee: $|E'| \geq |E(G)|/2 + |V(G)|/6$

Step 2: Layering

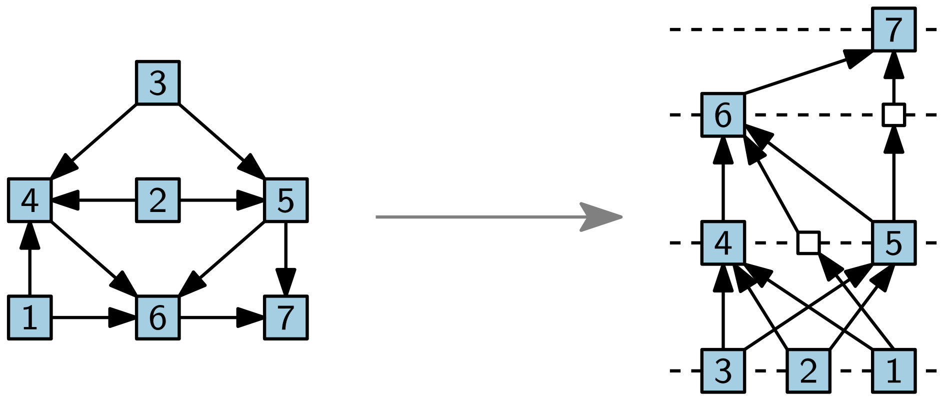


Step 2: Layering



Problem.

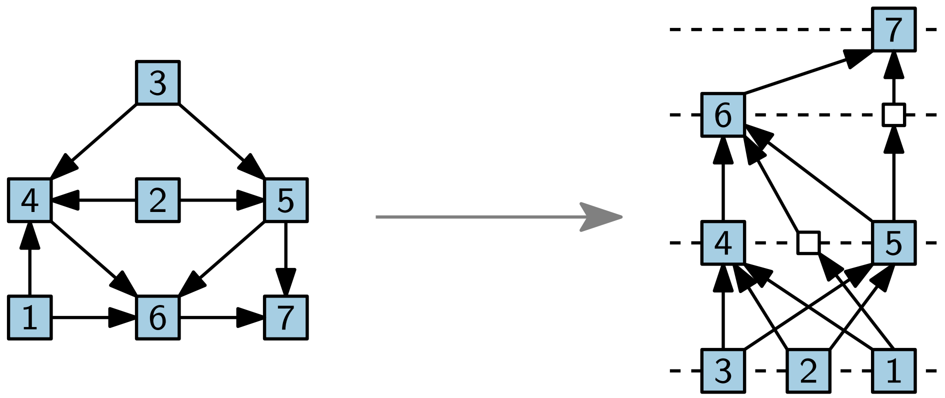
Step 2: Layering



Problem.

- Input: Acyclic digraph G.

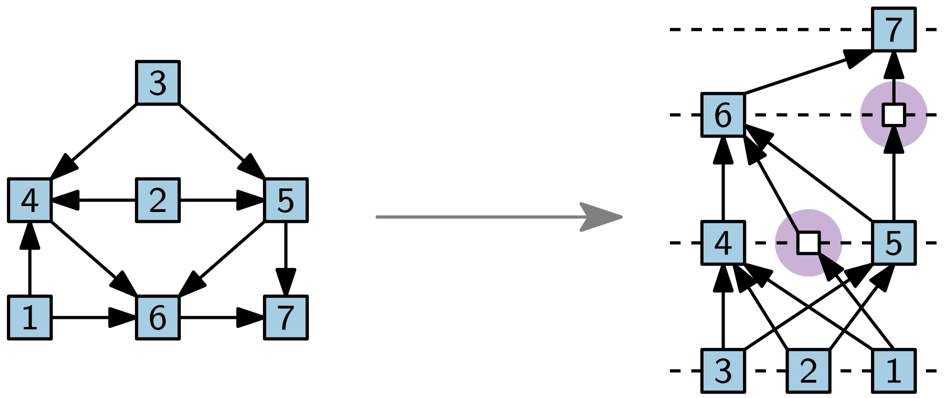
Step 2: Layering



Problem.

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

Step 2: Layering

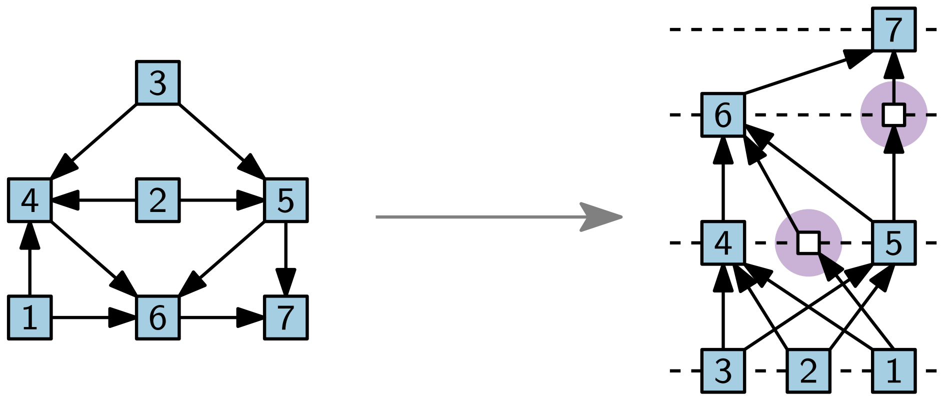


Whenever an edge spans across a layer, we insert a *dummy vertex*.

Problem.

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

Step 2: Layering



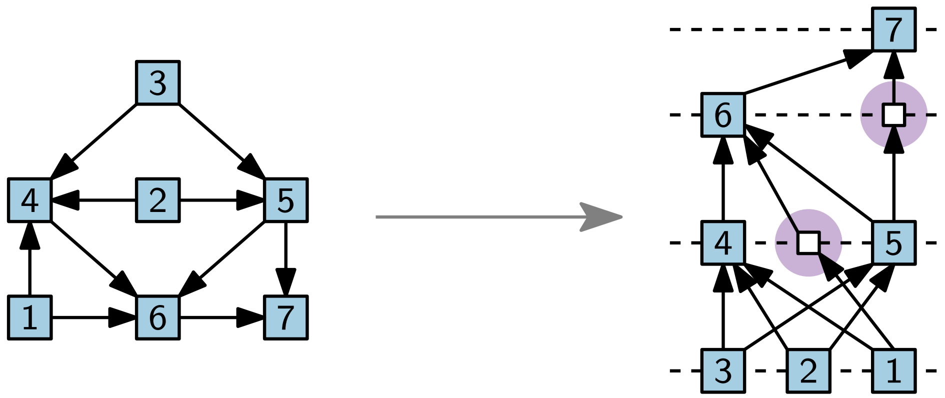
Whenever an edge spans across a layer, we insert a *dummy vertex*.

Problem.

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

Objective is to *minimize . . .*

Step 2: Layering



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

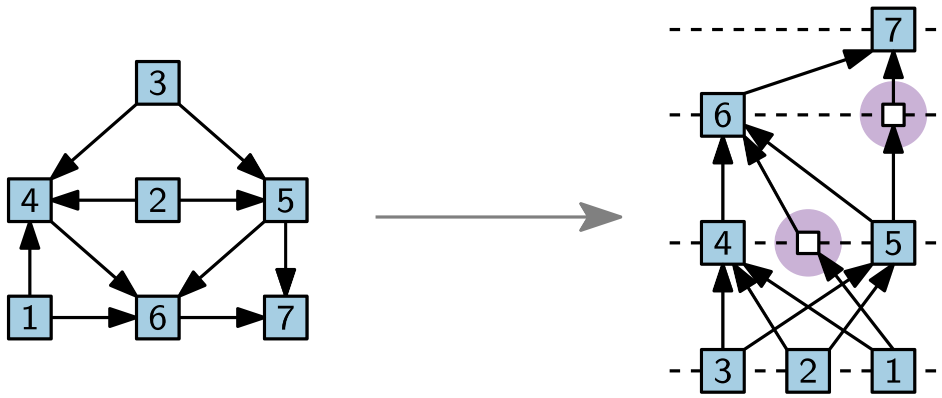
Problem.

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

Objective is to *minimize . . .*

- number of layers,

Step 2: Layering



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

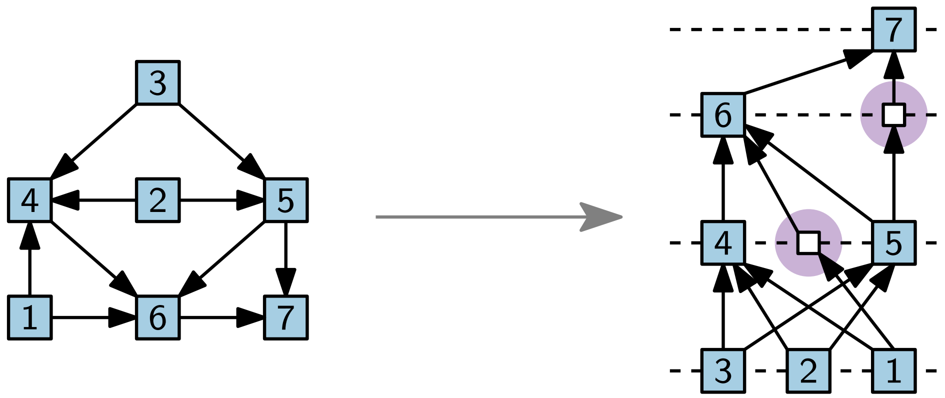
Problem.

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

Objective is to *minimize* . . .

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

Step 2: Layering



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

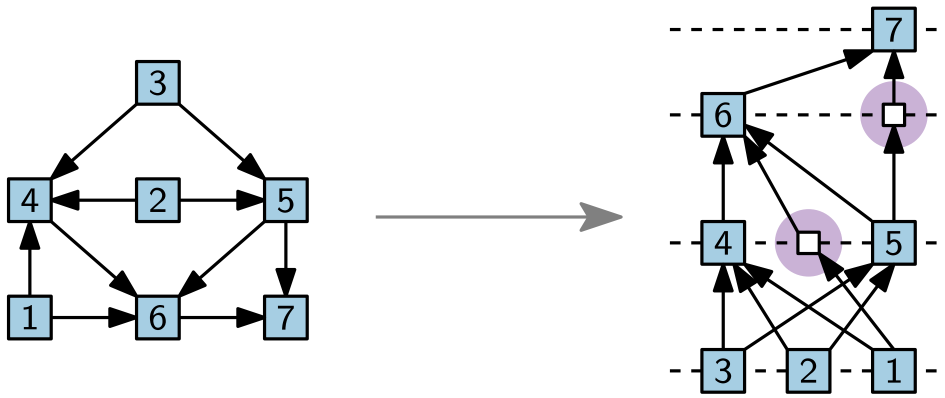
Problem.

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

Objective is to *minimize . . .*

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

Step 2: Layering



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

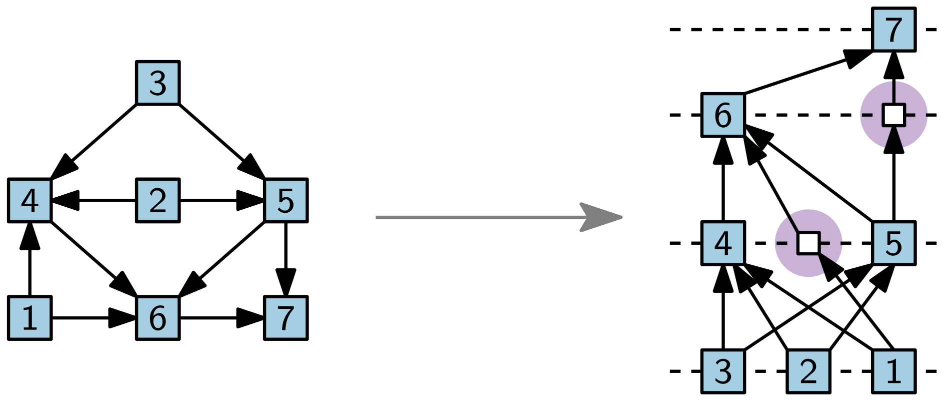
Problem.

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

Objective is to *minimize . . .*

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

Step 2: Layering



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

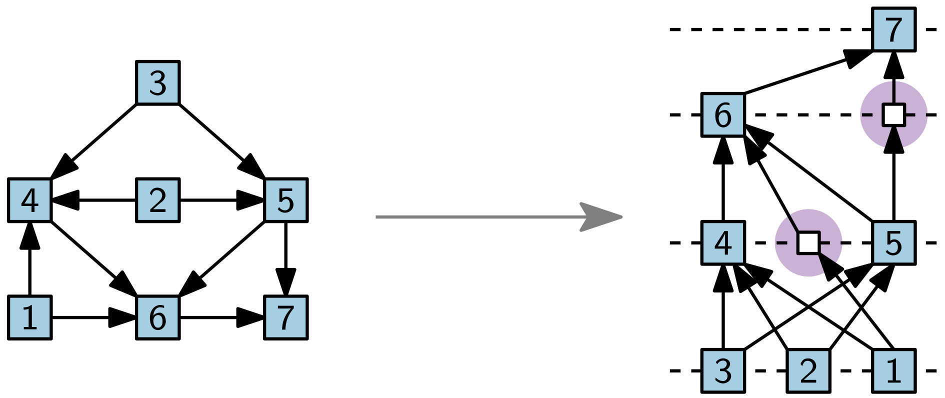
Problem.

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

Objective is to *minimize ...*

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

Step 2: Layering



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

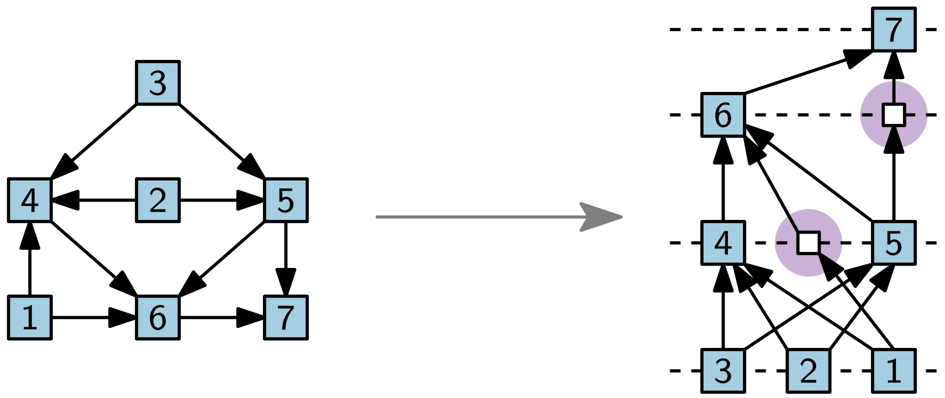
Problem.

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

Objective is to *minimize . . .*

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

Step 2: Layering



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

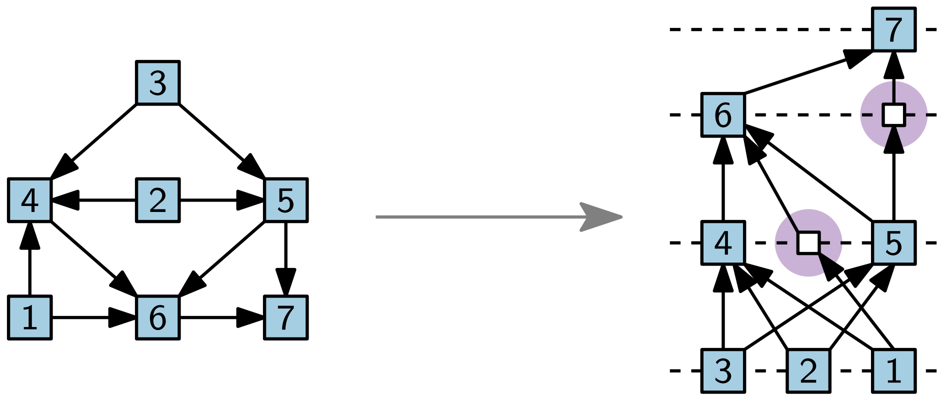
Problem.

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

Objective is to *minimize ...*

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

Step 2: Layering



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

Problem.

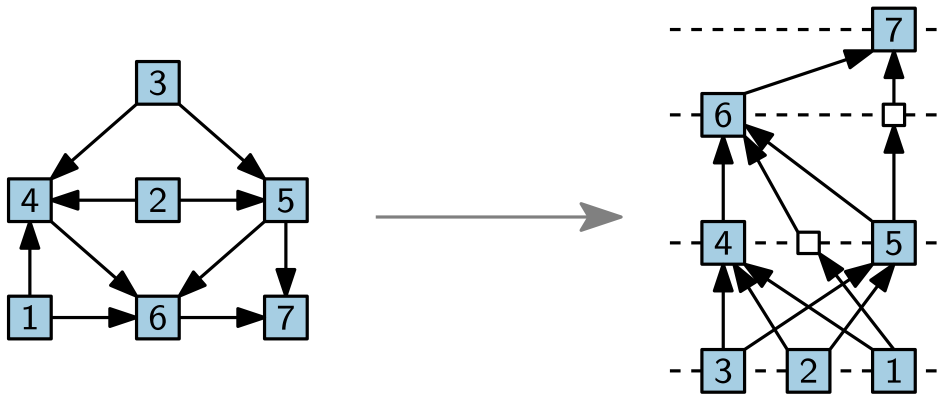
- Input: Acyclic digraph G.
- Output: Layering $y\colon V(G) \to \{1, \ldots, n\}$, such that, for every $(u, v) \in E(G)$, $y(u) < y(v)$.

Objective is to *minimize* ...

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

Minimize Number of Layers

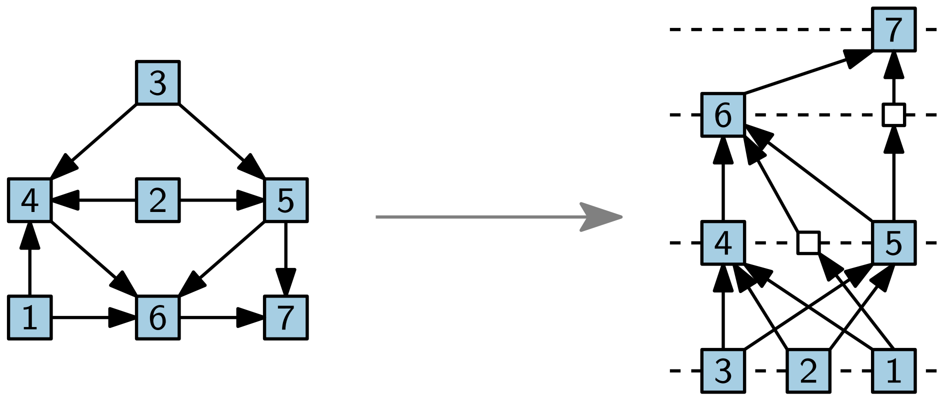
Algorithm.



Minimize Number of Layers

Algorithm.

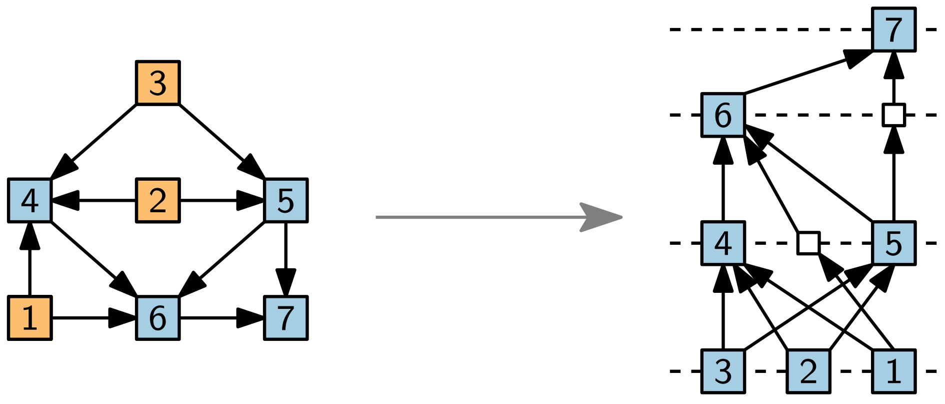
- for each source q,
  set $y(q) := 1$



Minimize Number of Layers

Algorithm.

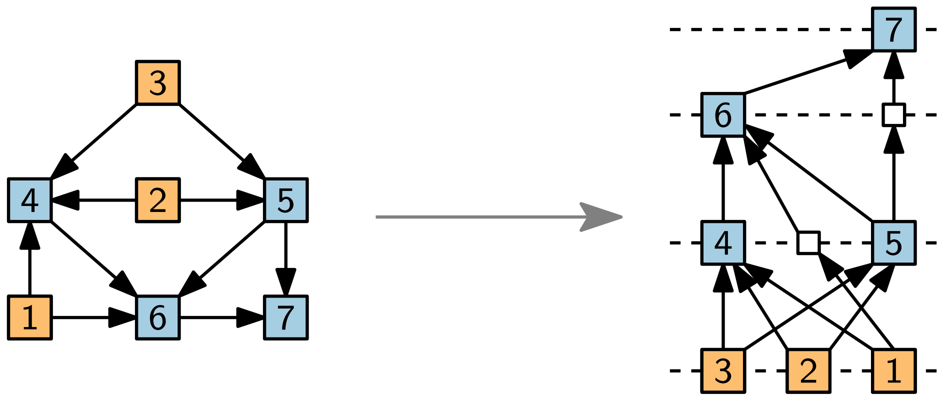
- for each source q,
  set $y(q) := 1$



Minimize Number of Layers

Algorithm.

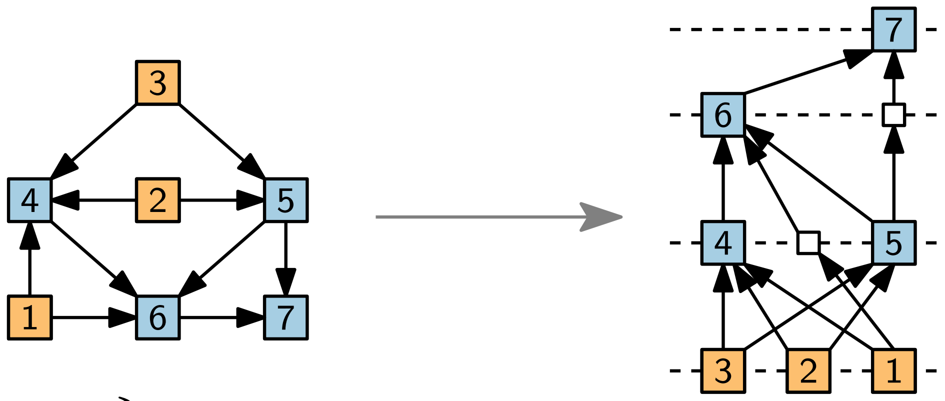
- for each source q,
  set $y(q) := 1$



Minimize Number of Layers

Algorithm.

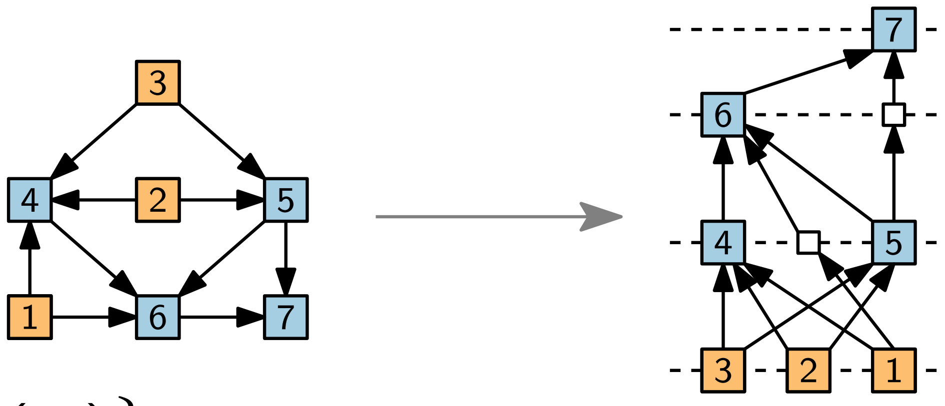
- for each source q,
  set $y(q) := 1$
- for each non-source v,
  set $y(v) := \max\left\{y(u) \mid (u, v) \in E(G)\right\} + 1$



Minimize Number of Layers

Algorithm.

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



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

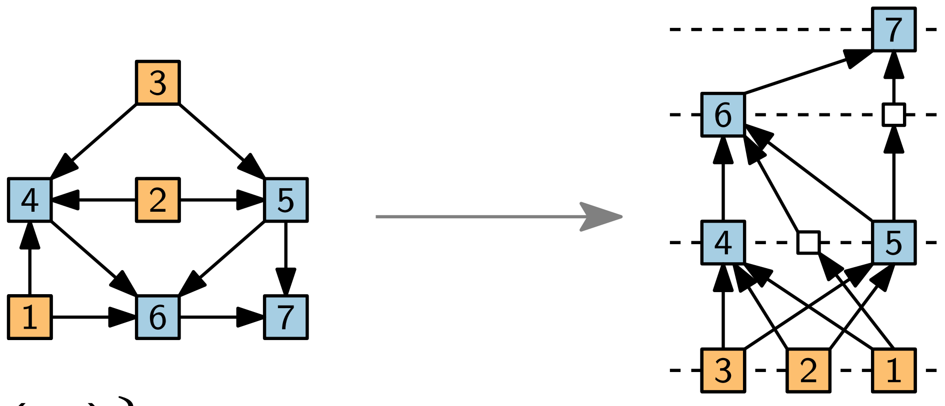
Observation.

- $y(v)$ is

Minimize Number of Layers

Algorithm.

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



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

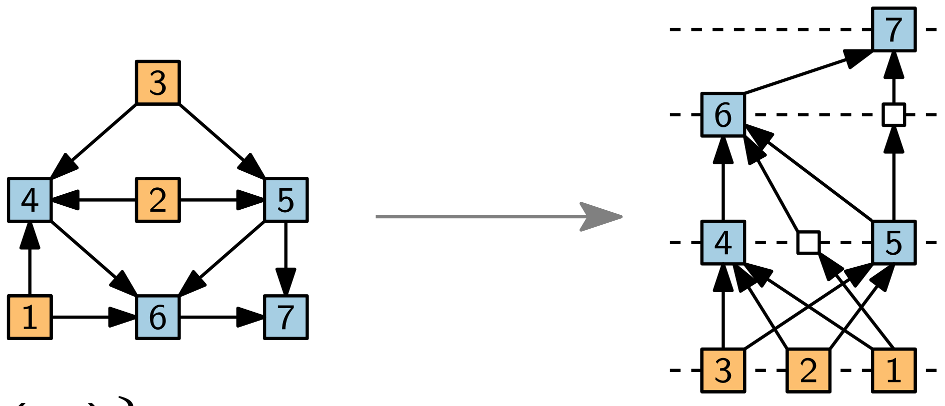
Observation.

- $y(v)$ is the 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 $y(v) := \max\left\{y(u) \mid (u,v) \in E(G)\right\} + 1$



Observation.

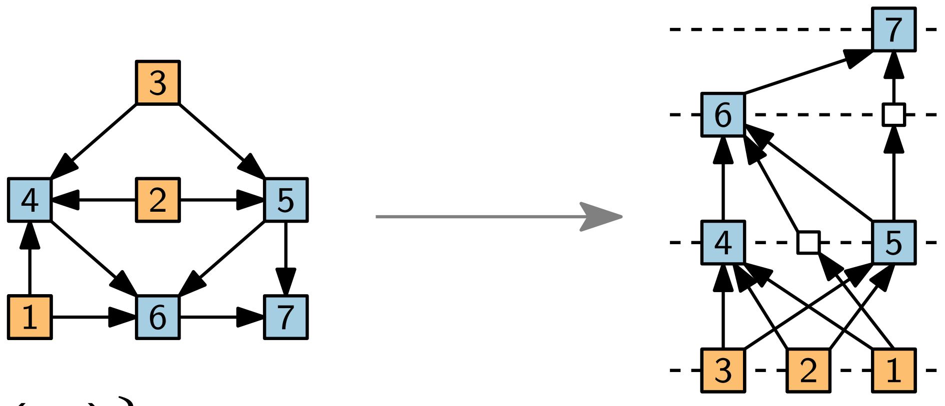
- $y(v)$ is the length of the longest path from a source to v plus 1.

  . . . which is optimal!

Minimize Number of Layers

Algorithm.

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



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

Observation.

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

  ...which is optimal!

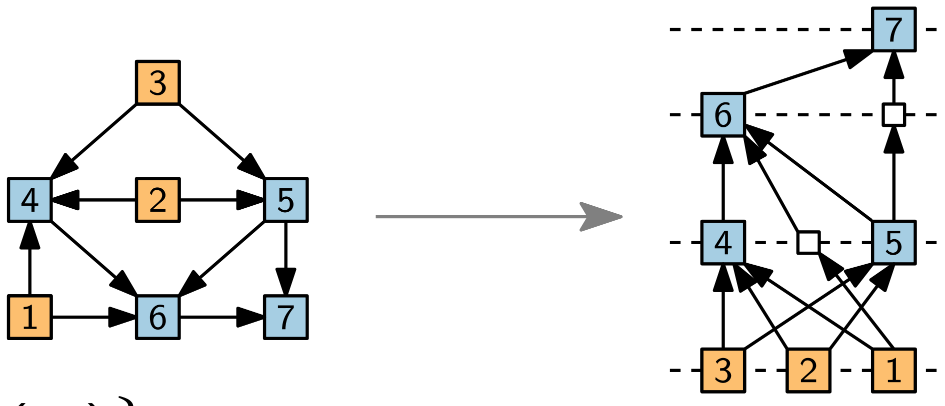
- Can be implemented in linear time, for example, using a recursive algorithm.

Minimize Number of Layers

Algorithm.

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

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



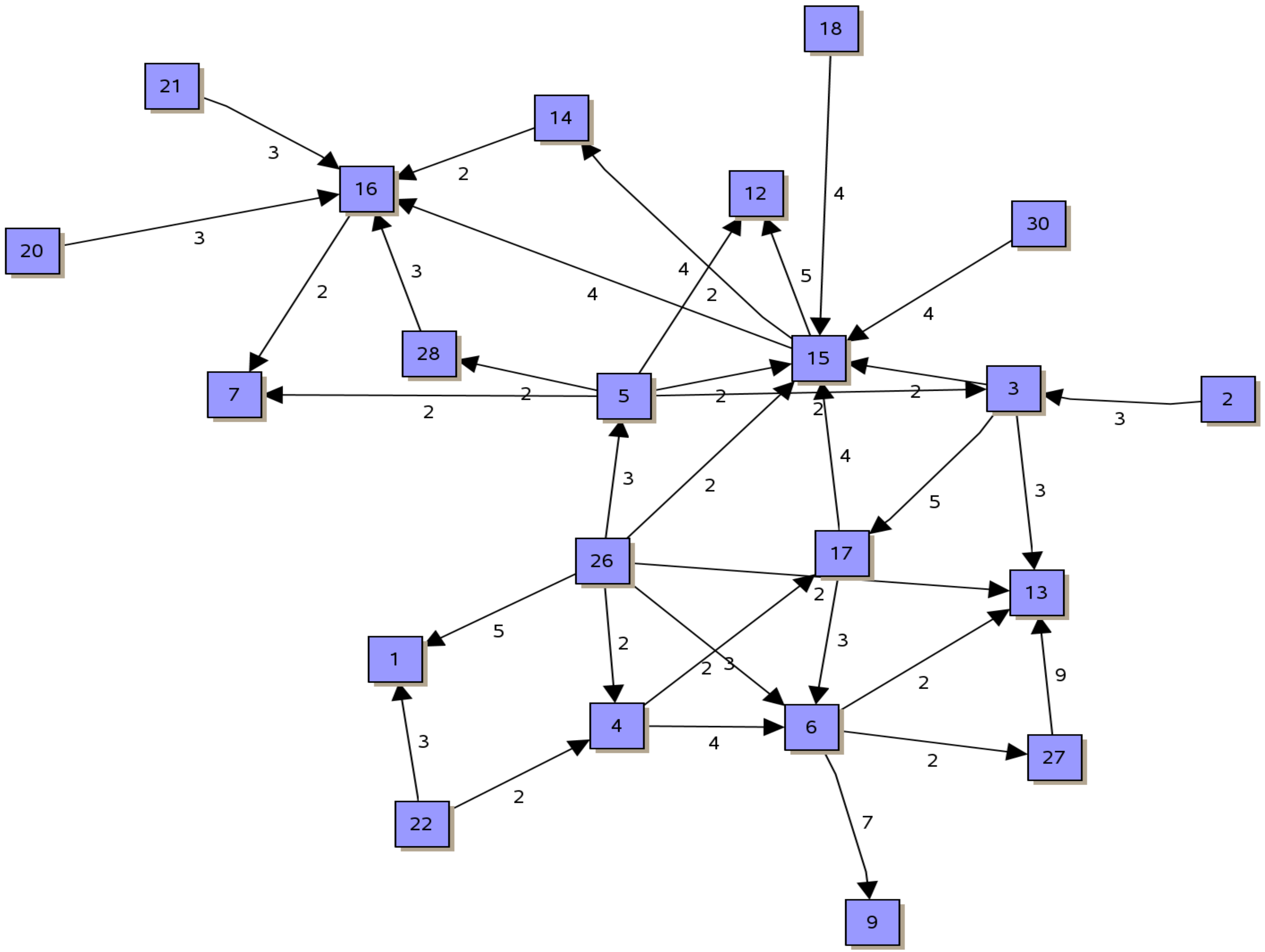
Observation.

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

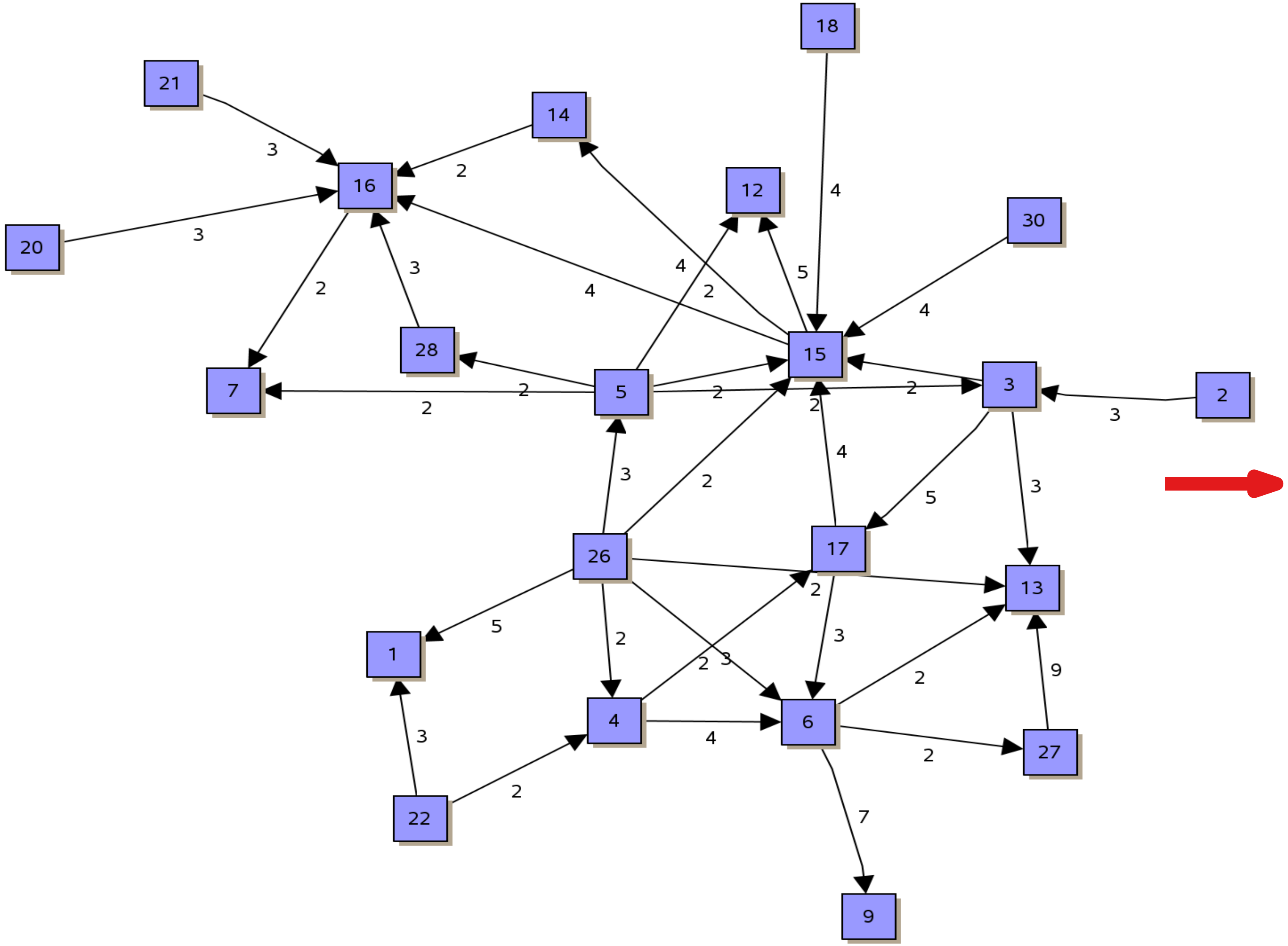
  . . . which is optimal!

- Can be implemented in linear time, for example, using a recursive algorithm.
- Closely related to topological sorting.

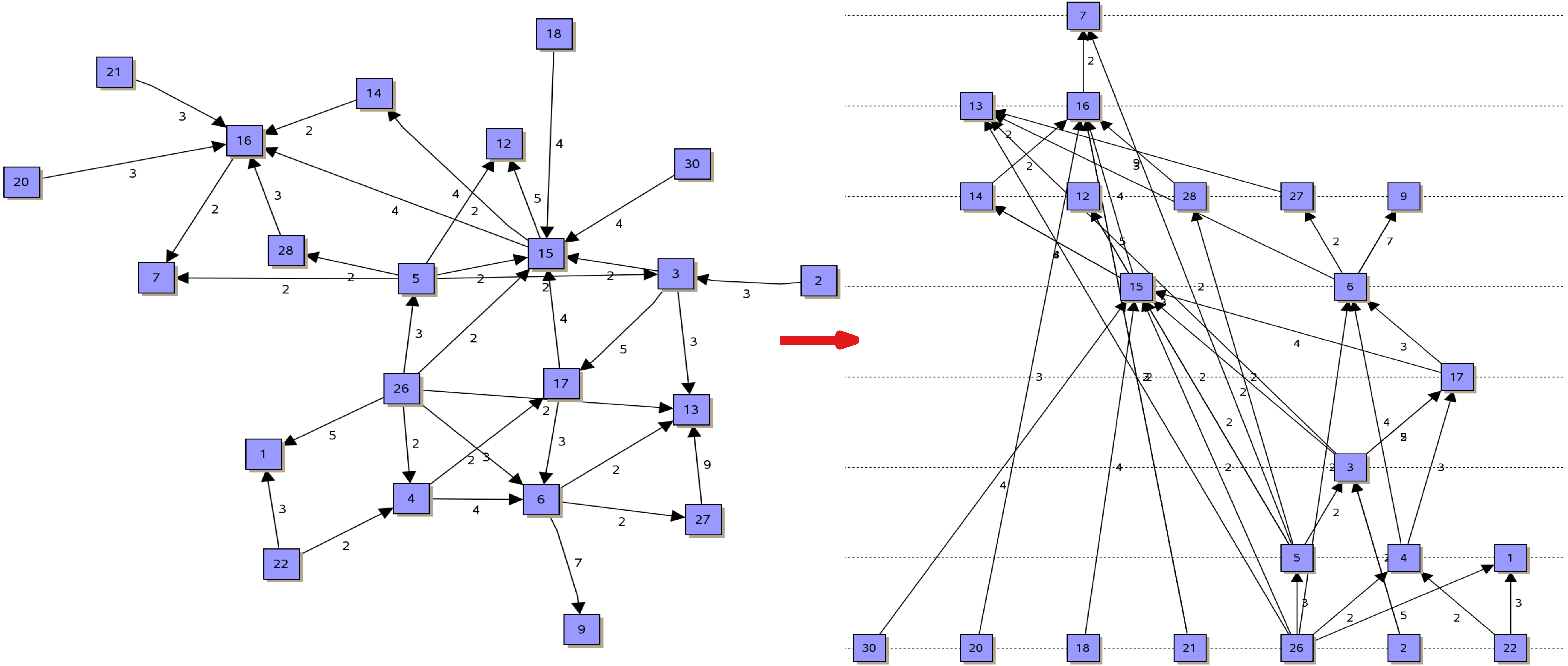
Example



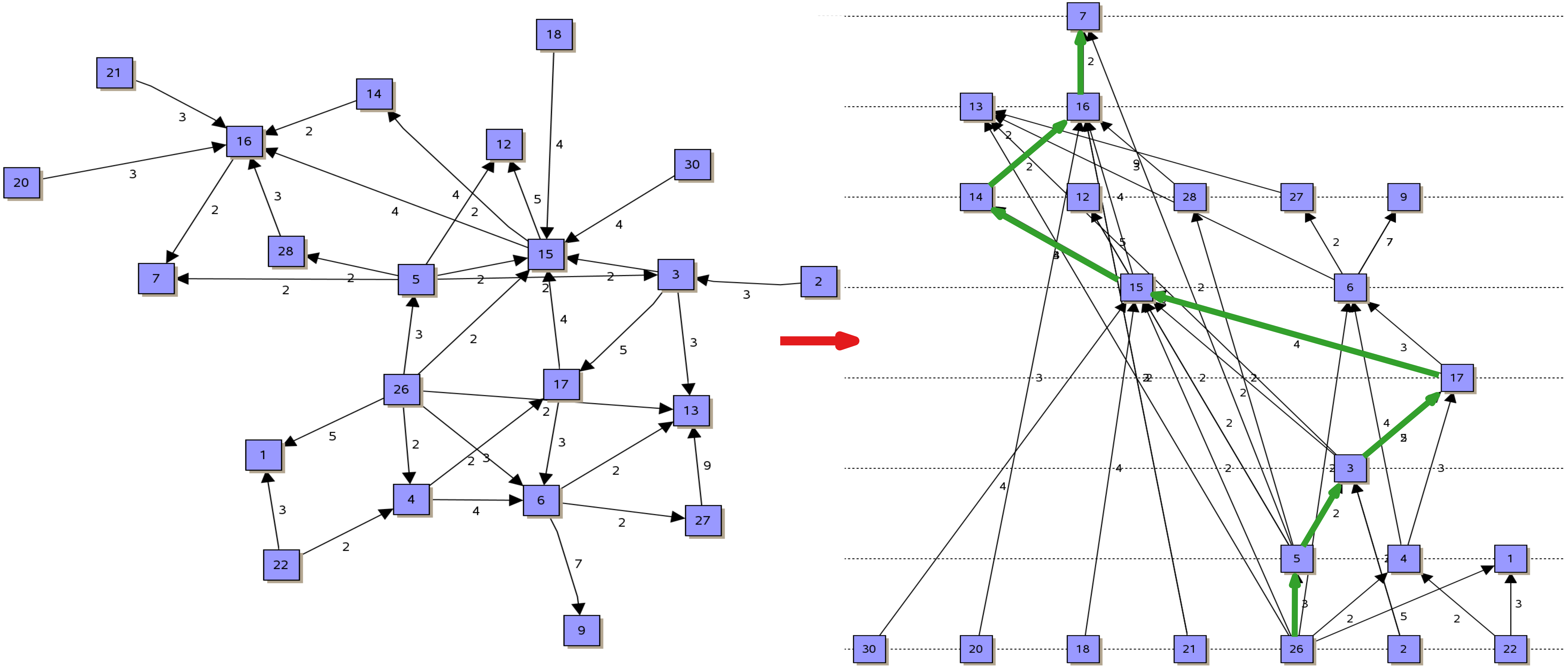
Example


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

Example



Example



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:

Minimize $\sum_{(u,v)\in E(G)}(y(v) - y(u))$

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

Minimize $\sum_{(u,v)\in E(G)}(y(v) - y(u))$

subject to

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

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

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

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

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

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

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

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

One can show that:

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

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

One can show that:

- Constraint matrix is **totally unimodular** (every square submatrix has det in $\{-1,0,1\}$).

Minimize Total Edge Length – ILP

Can be formulated as an integer linear program:

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

One can show that:

- Constraint matrix is **totally unimodular** (every square submatrix has det in $\{-1,0,1\}$).
  $\Rightarrow$ Extreme point solutions of the LP relaxation (ILP without $y(v)\in\mathbb{Z}$) are integer.

Minimize Total Edge Length – ILP

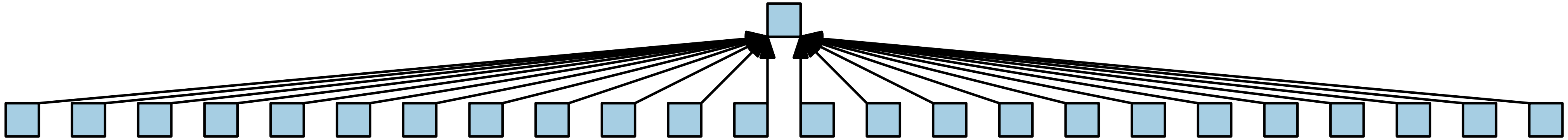
Can be formulated as an integer linear program:

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

One can show that:

- Constraint matrix is **totally unimodular** (every square submatrix has det in $\{-1,0,1\}$).
  $\Rightarrow$ Extreme point solutions of the LP relaxation (ILP without $y(v)\in\mathbb{Z}$) are integer.
- The total edge length can be minimized in polynomial time.

Width



Drawings can be very wide.

Narrower Layer Assignment

Problem: Layering with a given maximum width.

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – each layer contains at most W elements, and
  – the number of layers is minimized.

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – each layer contains at most W elements, and
  – the number of layers is minimized.

Problem: Precedence-Constrained Multi-Processor Scheduling.

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – 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 processing time, W identical machines,
  partial ordering $<$ on the jobs.

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – 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 processing time, W identical machines, partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ such that completion time (known as *makespan*) is minimized.

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – 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 processing time, W identical machines, partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ such that completion time (known as *makespan*) is minimized.

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – 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 processing time, W identical machines, partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ such that completion time (known as *makespan*) is minimized.

- NP-hard

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – 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 processing time, W identical machines, partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ such that completion time (known as *makespan*) is minimized.

- NP-hard, $(2 - 1/W)$-approximation

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – 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 processing time, W identical machines, partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ such that completion time (known as *makespan*) is minimized.

- NP-hard, $(2 - 1/W)$-approximation, no $(4/3 - \varepsilon)$-approximation ($W \geq 3$)

Narrower Layer Assignment

Problem: Layering with a given maximum width.

- Input: Acyclic digraph G, width $W > 0$
- Output: Assignment of the vertices of G to layers such that
  – the assignment is a layering,
  – 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 processing time, W identical machines, partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ such that completion time (known as *makespan*) is minimized.
- NP-hard, $\boxed{(2 - 1/W)\text{-approximation}}$, no $(4/3 - \varepsilon)$-approximation ($W \geq 3$)

Approximating Precedence-Const. Multi-Processor Scheduling

- Jobs stored in a list L, which is topologically sorted.

Approximating Precedence-Const. Multi-Processor Scheduling

- Jobs stored in a list L, which is topologically sorted.
- A job in L is *available* if all its predecessors have been scheduled.

Approximating Precedence-Const. Multi-Processor Scheduling

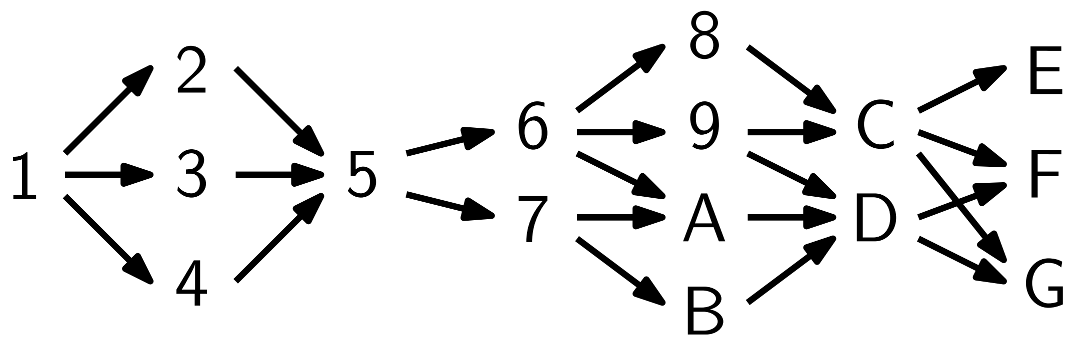
- Jobs stored in a list L, which is topologically sorted.
- A job in L is *available* if all its predecessors have been scheduled.
- For each point in time $t = 1, 2, \ldots,$ we can schedule $\leq W$ available jobs.

Approximating Precedence-Const. Multi-Processor Scheduling

- Jobs stored in a list L, which is topologically sorted.
- A job in L is *available* if all its predecessors have been scheduled.
- For each point in 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.

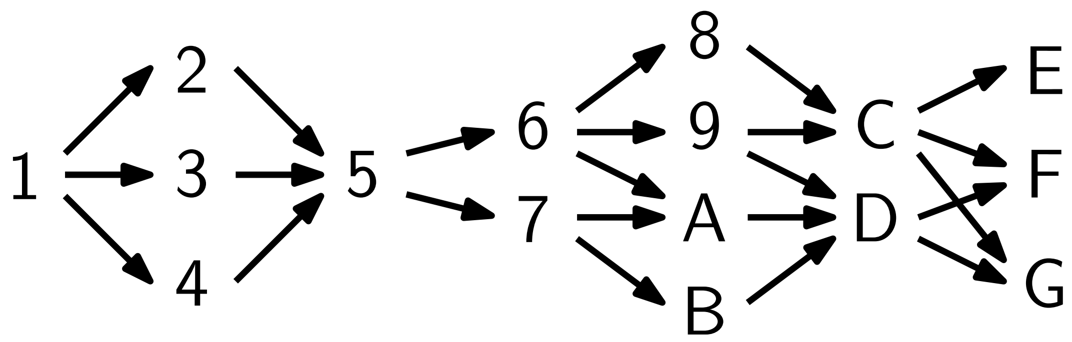
Approximating Precedence-Const. Multi-Processor Scheduling

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



Approximating Precedence-Const. Multi-Processor Scheduling

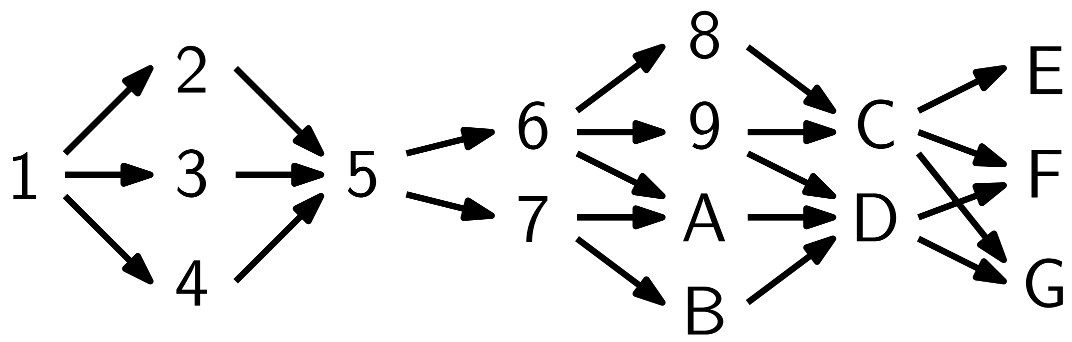
Input: Precedence graph (divided into layers of arbitrary width)



Number of machines is $W = 2$.

Approximating Precedence-Const. Multi-Processor Scheduling

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

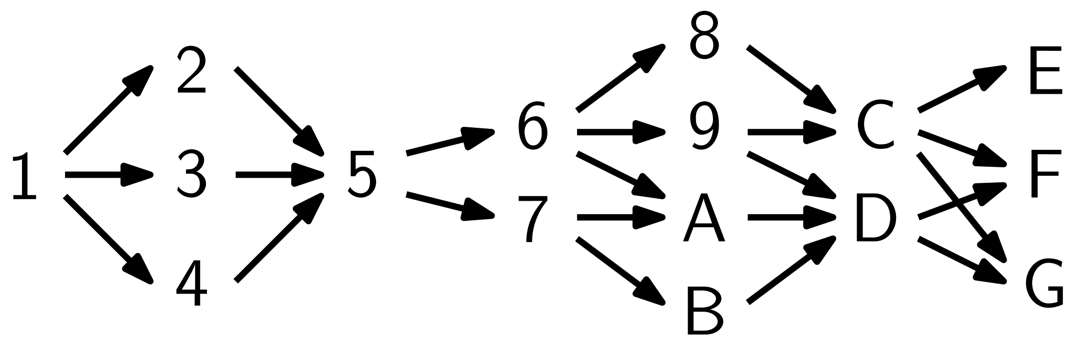


Number of machines is $W = 2$.

Output: Schedule

Approximating Precedence-Const. Multi-Processor Scheduling

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



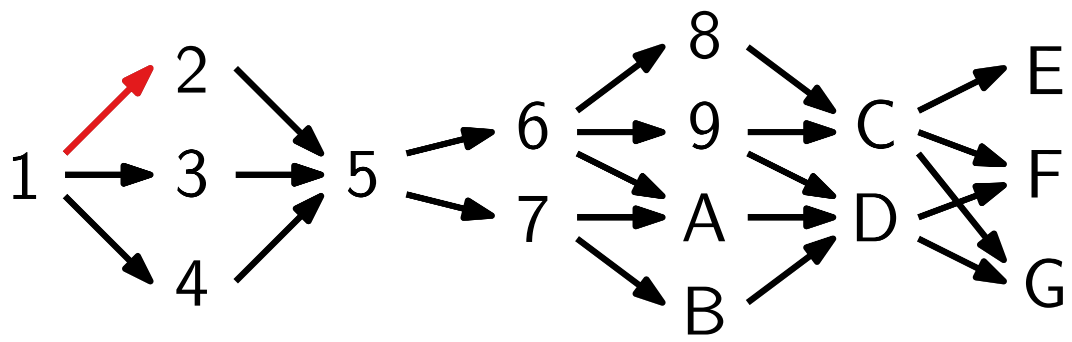
Number of machines is $W = 2$.

Output: Schedule

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

Approximating Precedence-Const. Multi-Processor Scheduling

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



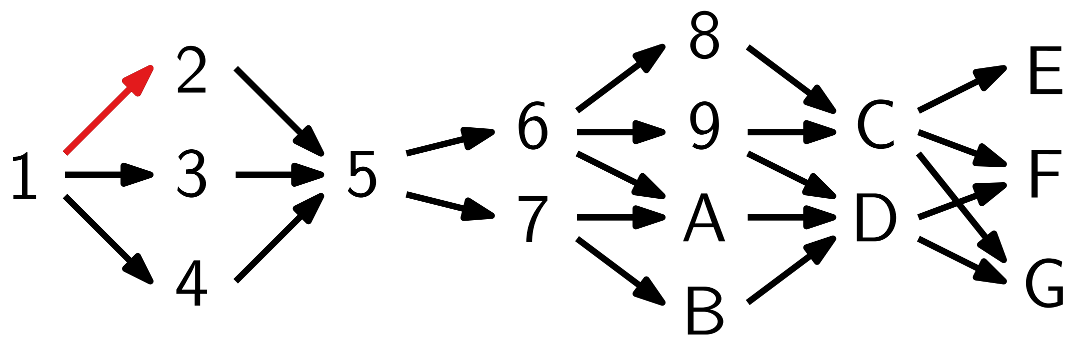
Number of machines is $W = 2$.

Output: Schedule

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

Approximating Precedence-Const. Multi-Processor Scheduling

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



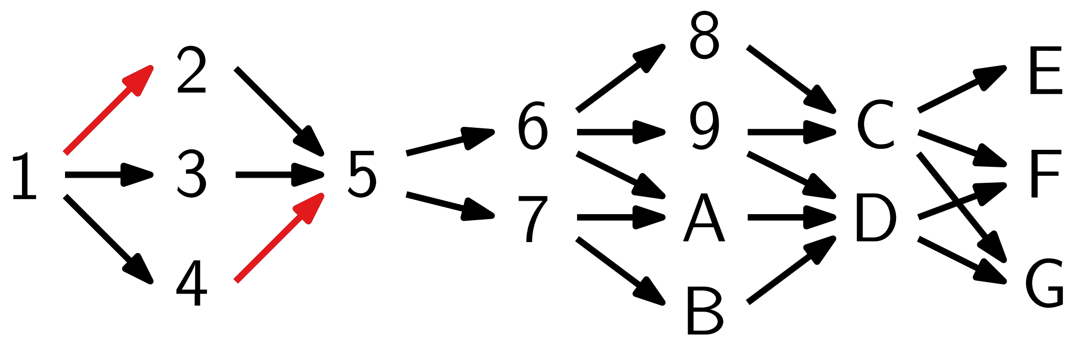
Number of machines is $W = 2$.

Output: Schedule

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

Approximating Precedence-Const. Multi-Processor Scheduling

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



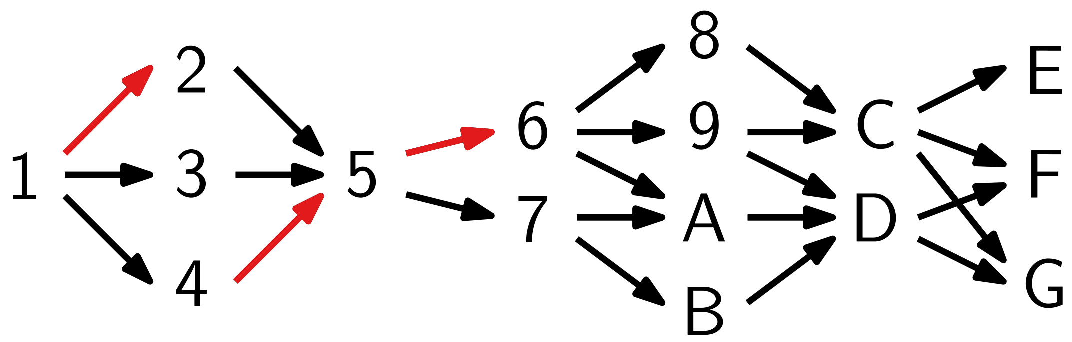
Number of machines is $W = 2$.

Output: Schedule

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

Approximating Precedence-Const. Multi-Processor Scheduling

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



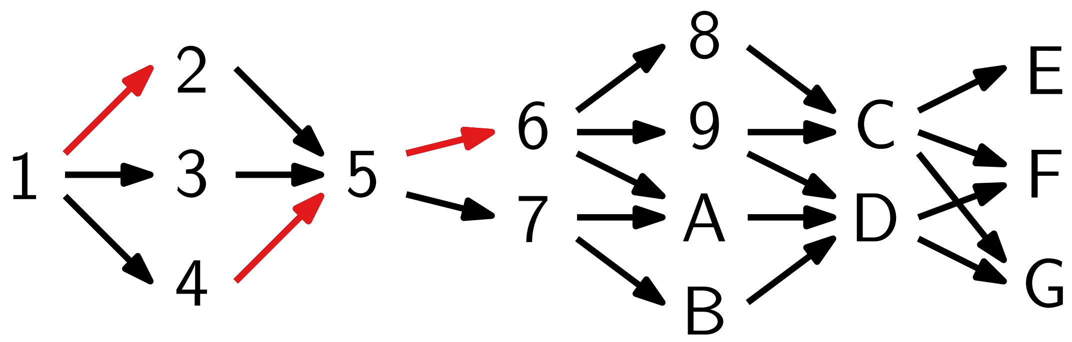
Number of machines is $W = 2$.

Output: Schedule

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

Approximating Precedence-Const. Multi-Processor Scheduling

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



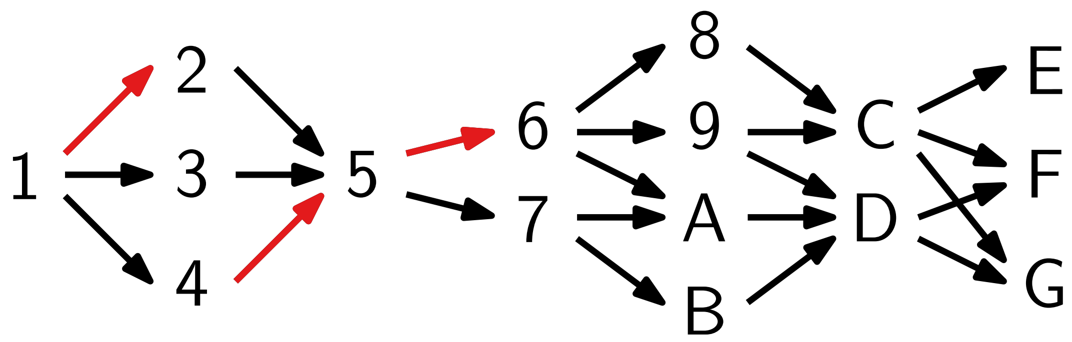
Number of machines is $W = 2$.

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 Precedence-Const. Multi-Processor Scheduling

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



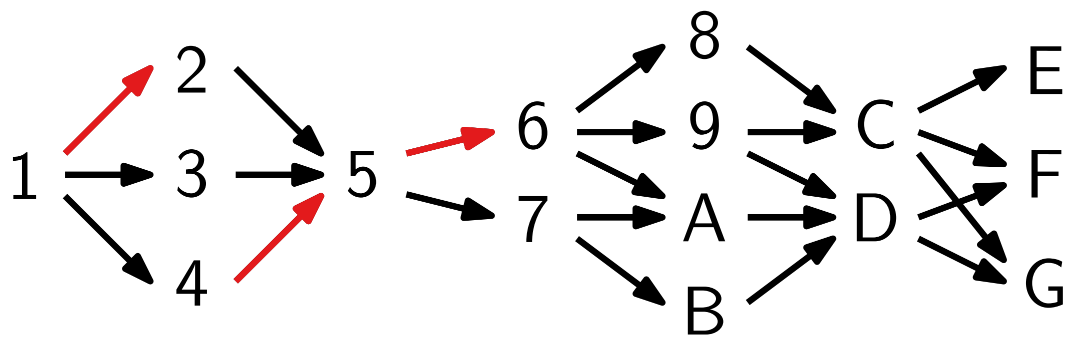
Number of machines is $W = 2$.

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 Precedence-Const. Multi-Processor Scheduling

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



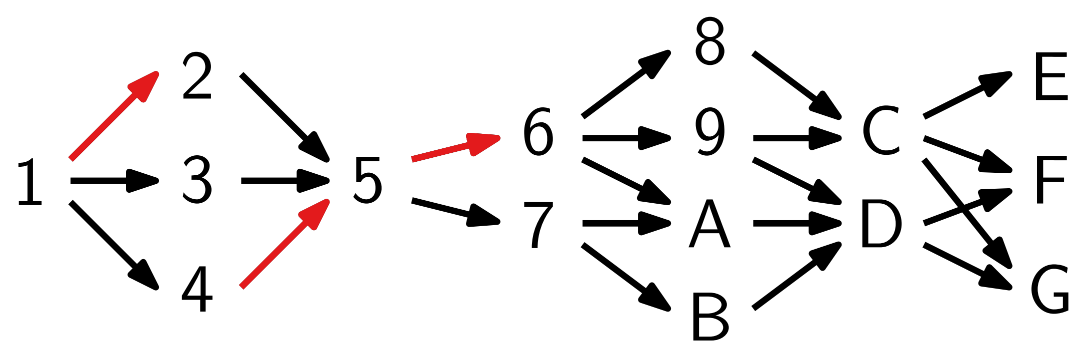
Number of machines is $W = 2$.

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 Precedence-Const. Multi-Processor Scheduling

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



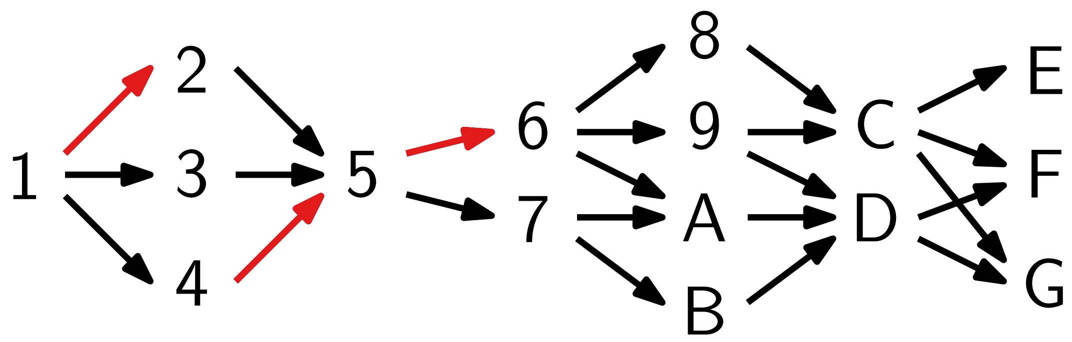
Number of machines is $W = 2$.

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 Precedence-Const. Multi-Processor Scheduling

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



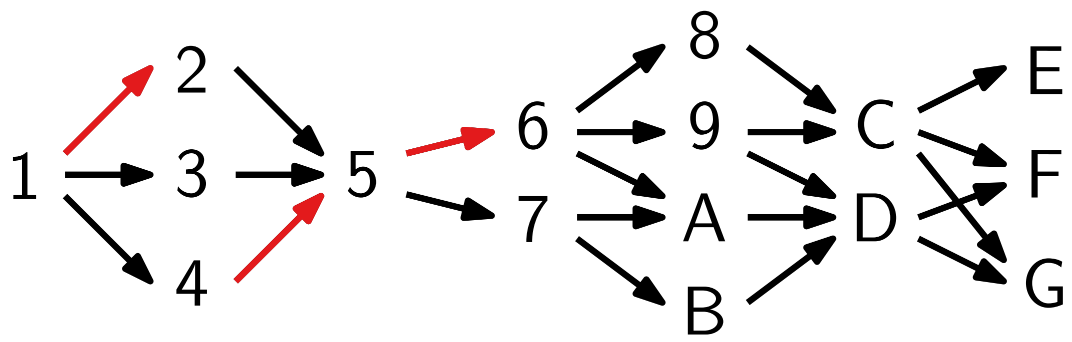
Number of machines is $W = 2$.

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 Precedence-Const. Multi-Processor Scheduling

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



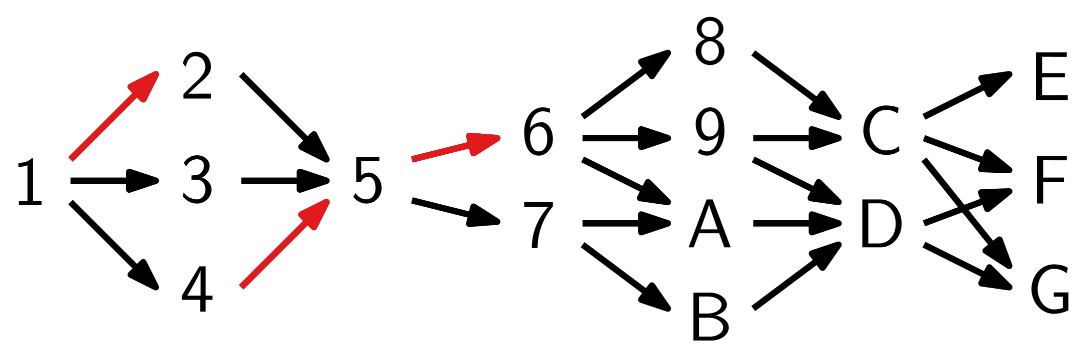
Number of machines is $W = 2$.

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 Precedence-Const. Multi-Processor Scheduling

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



Number of machines is $W = 2$.

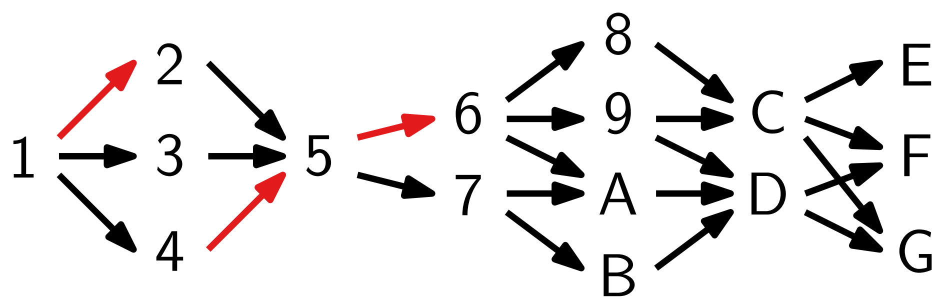
Output: Schedule

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 $G_<$



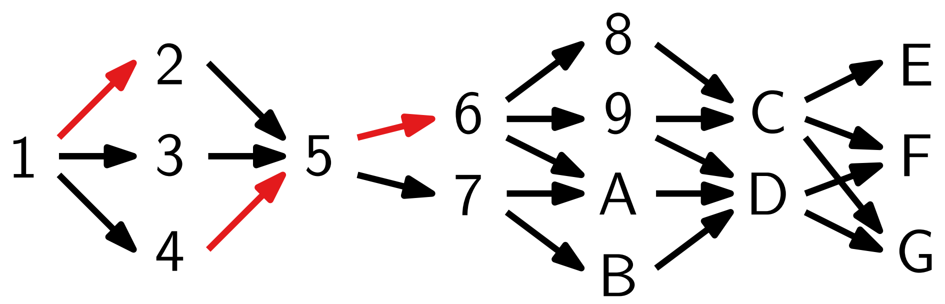
Schedule

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

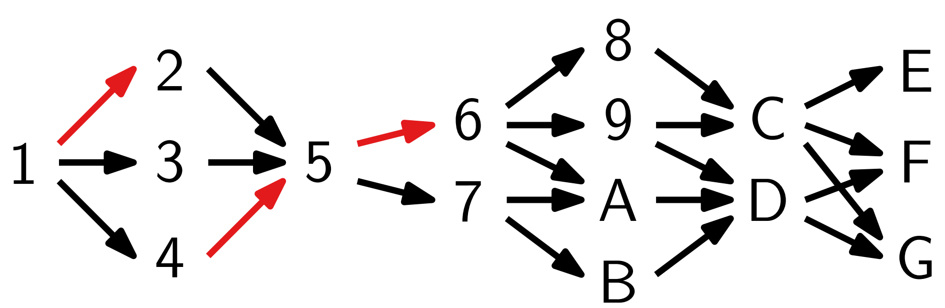
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„The art of the lower bound“

OPT $\geq$

Approximating PCMPS – Analysis for $W = 2$

Precedence graph $G_<$



Schedule

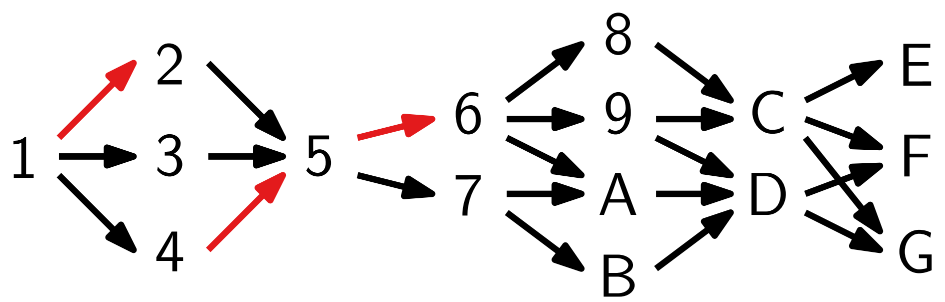
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„The art of the lower bound“

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

Approximating PCMPS – Analysis for $W = 2$

Precedence graph $G_<$



Schedule

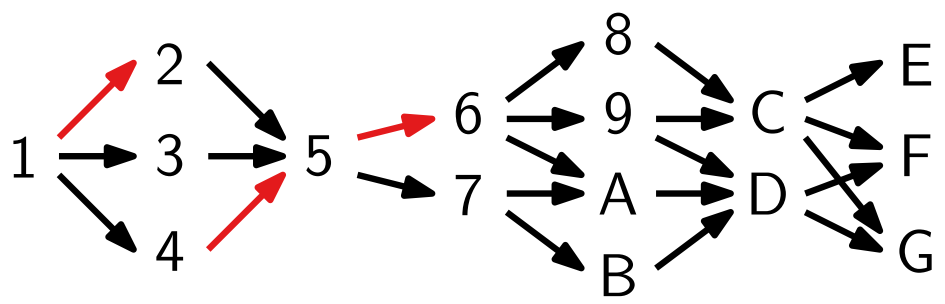
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„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

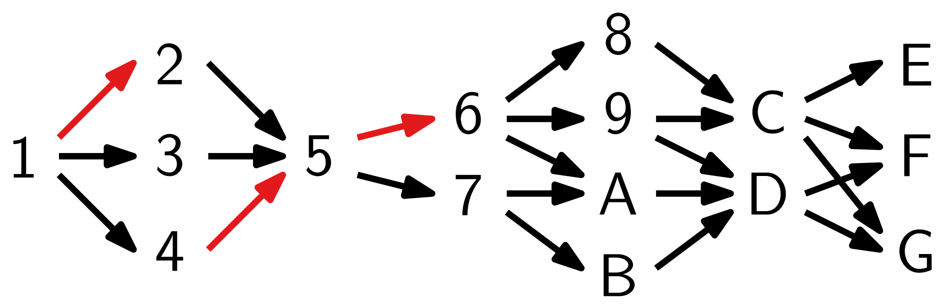
M_1	1	2	4	5	6	8	A	C	E	G
M_2	–	3	–	–	7	9	B	D	F	–
t	1	2	3	4	5	6	7	8	9	10

„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

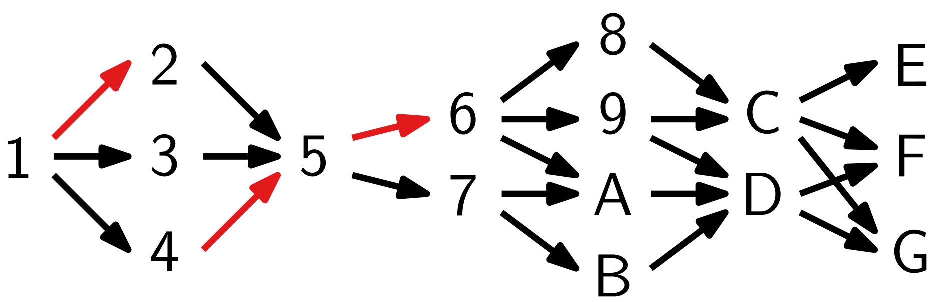
„The art of the lower bound“

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“

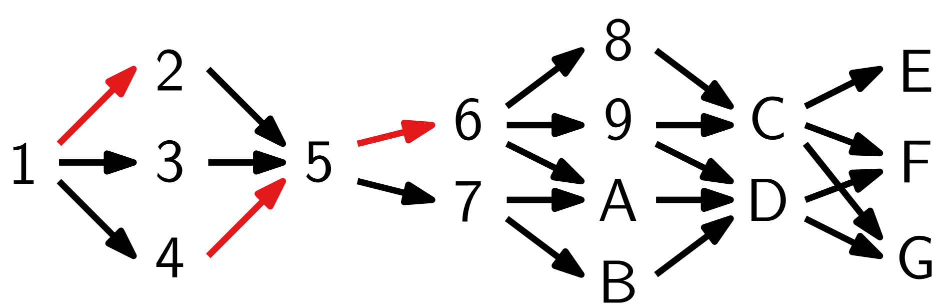
OPT $\geq$ $\lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

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“

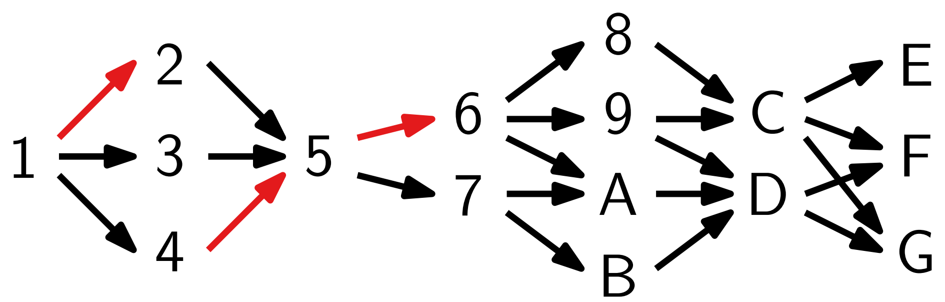
OPT $\geq \lceil n/2 \rceil$ and OPT $\geq \ell :=$ Number of layers of $G_<$ ($=$ length of longest path in $G_<$)

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 $G_<$)

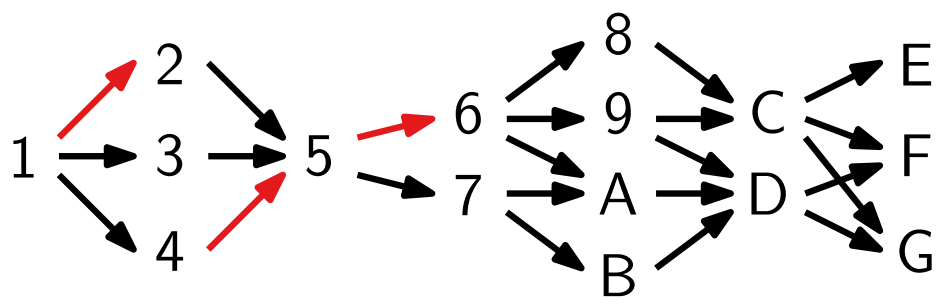
Goal: measure the quality of our algorithm using the lower bounds

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 $G_<$)

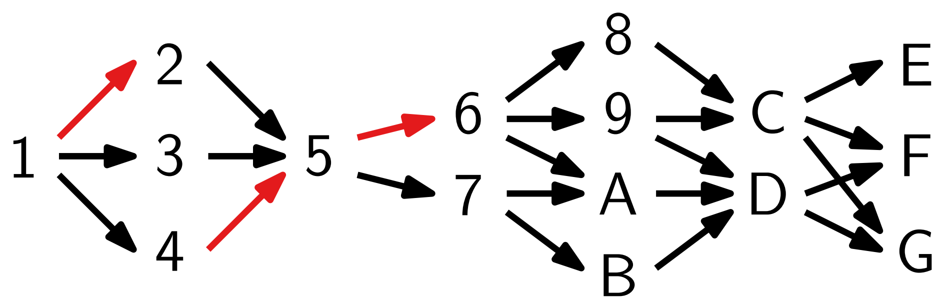
Goal: measure the quality of our algorithm using the lower bounds

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 $G_<$)

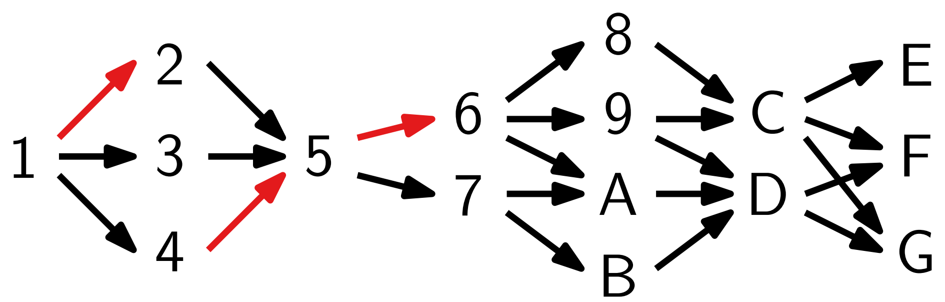
Goal: measure the quality of our algorithm using the lower bounds

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 $G_<$)

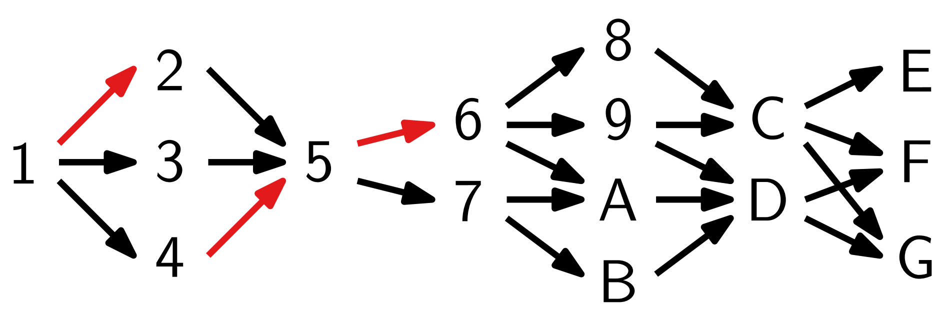
Goal: measure the quality of our algorithm using the lower bounds

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 $G_<$)

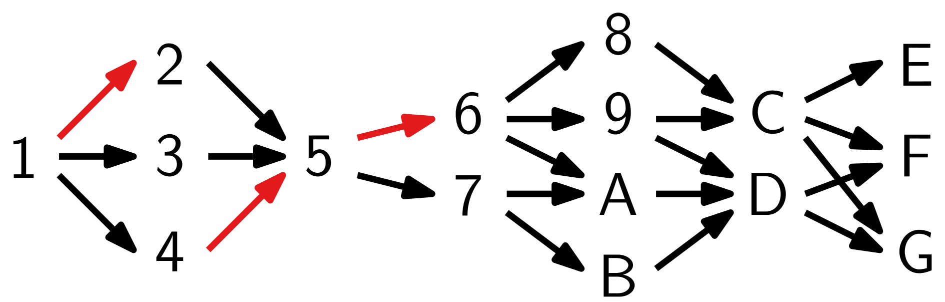
Goal: measure the quality of our algorithm using the lower bounds

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 $G_<$)

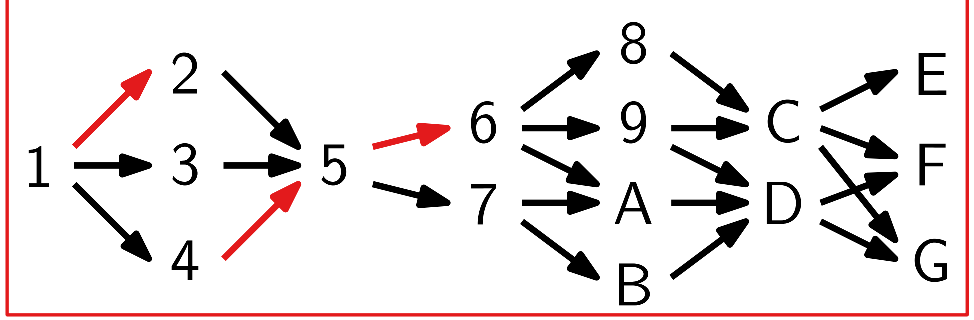
Goal: measure the quality of our algorithm using the lower bounds

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 $G_<$)

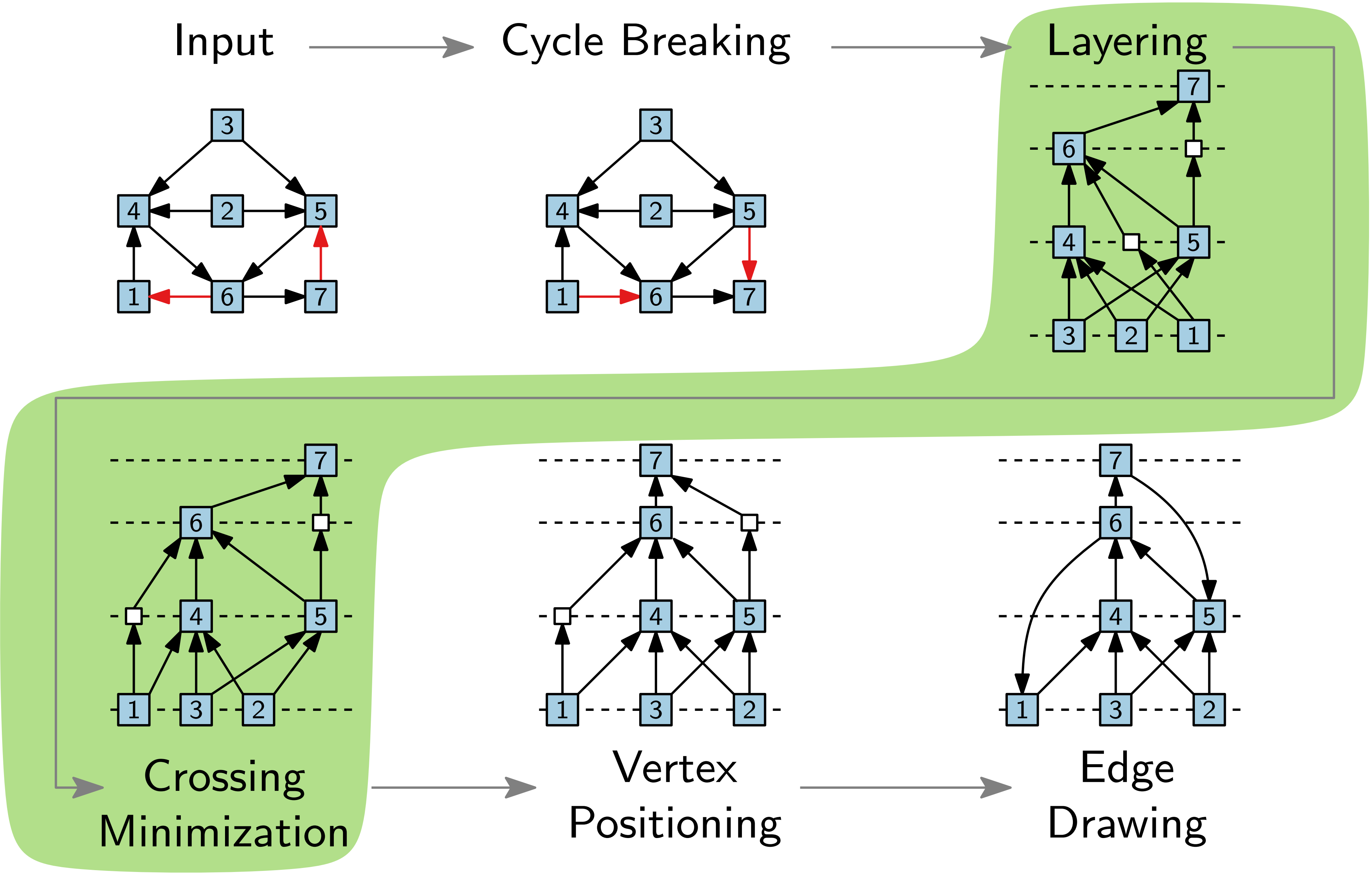
Goal: measure the quality of our algorithm using the lower bounds

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

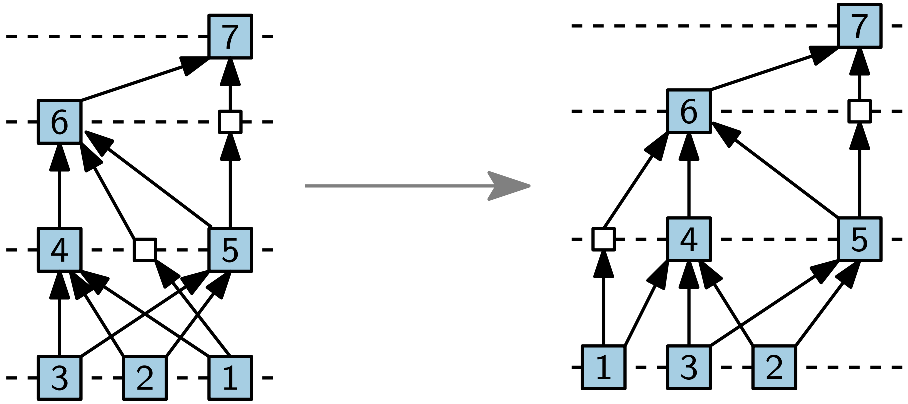
Bound. ALG $\leq \left\lceil \frac{n+\ell}{2} \right\rceil \approx \lceil n/2 \rceil + \ell/2 \leq 3/2 \cdot$ OPT

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

Step 3: Crossing Minimization

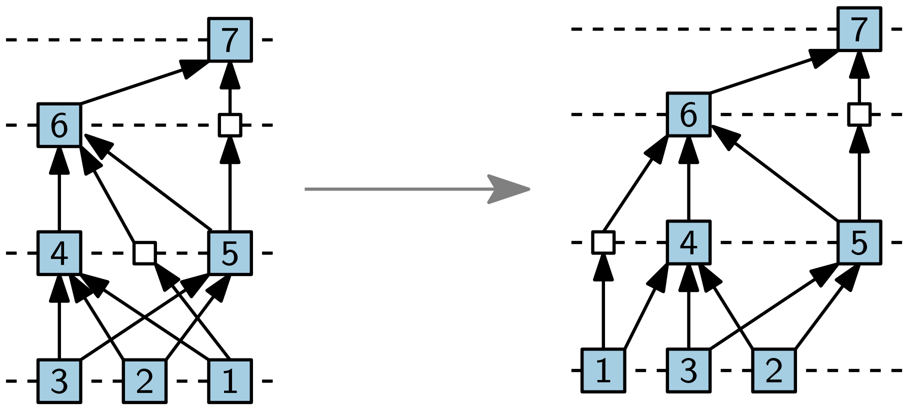


Step 3: Crossing Minimization



Problem.

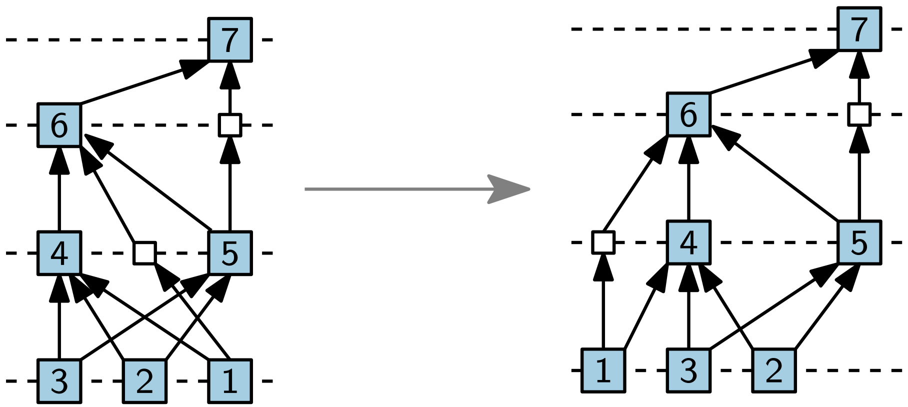
Step 3: Crossing Minimization



Problem.

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

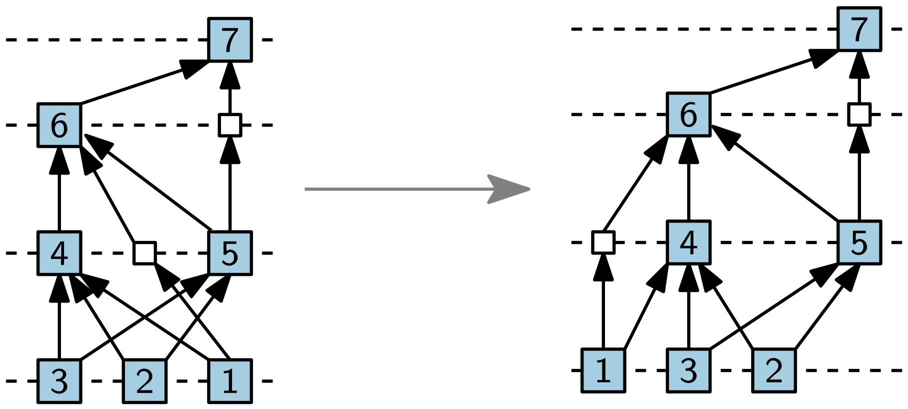
Step 3: Crossing Minimization



Problem.

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

Step 3: Crossing Minimization

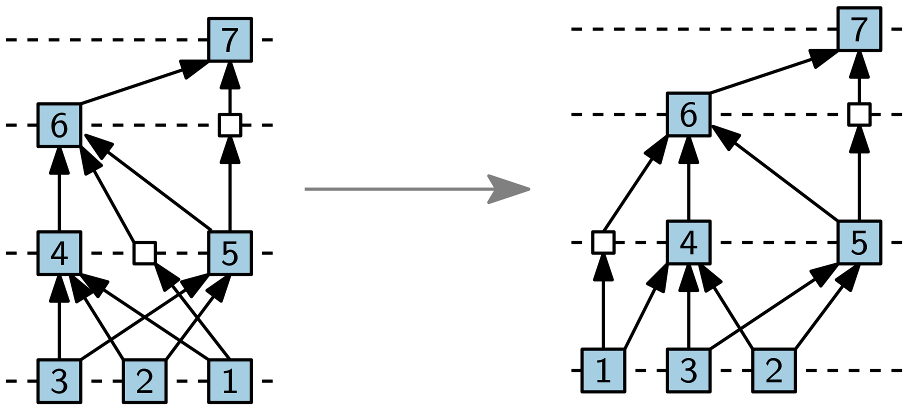


Problem.

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

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

Step 3: Crossing Minimization



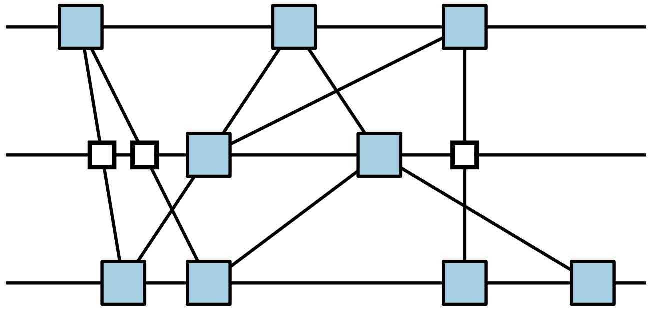
Problem.

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

- NP-hard, even for two layers. [Garey & Johnson '83]
- Hardly any approaches optimize over multiple layers. 

Iterative Crossing Reduction

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

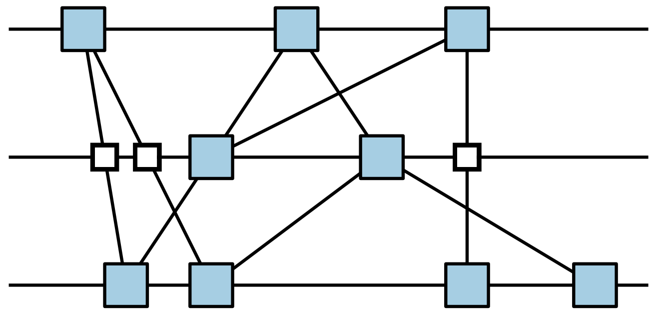


Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as "regular" vertices.

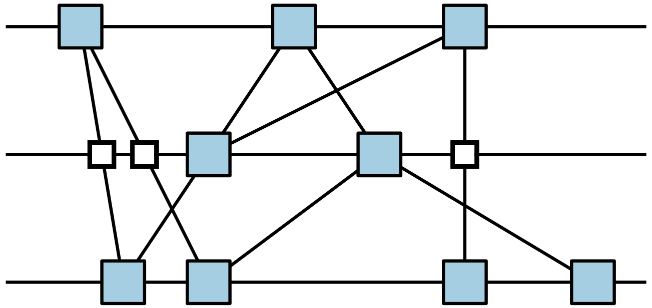


Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “regular” vertices.



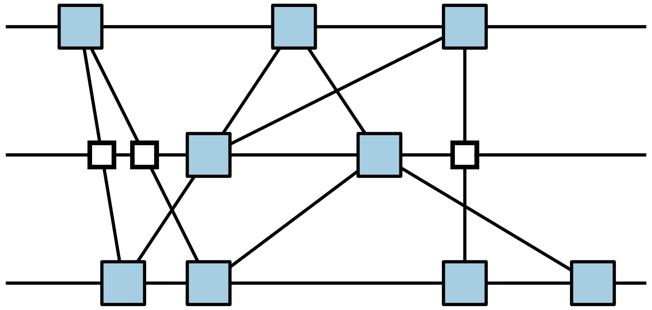
Algorithm scheme.

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “regular” vertices.



Algorithm scheme.

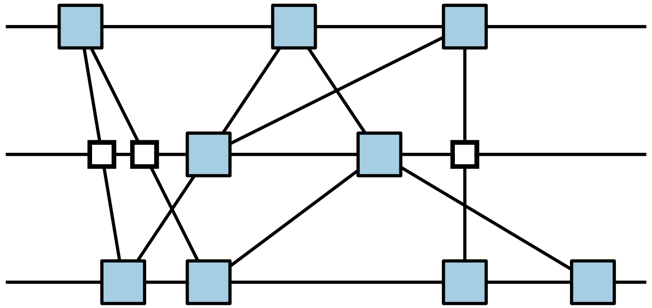
(1) choose a random permutation of L_1

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “regular” vertices.



Algorithm scheme.

(1) choose a random permutation of L_1

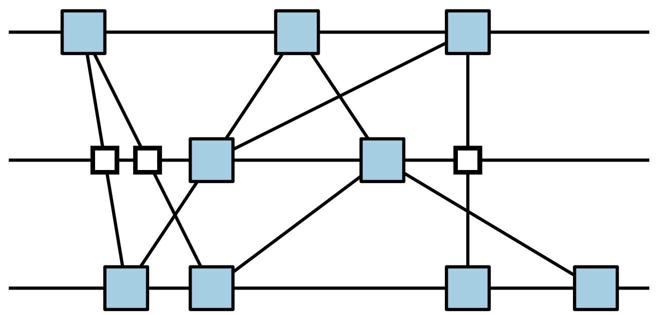
(2) iteratively consider pairs of adjacent layers (L_i, L_{i+1})

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “regular” vertices.



Algorithm scheme.

(1) choose a random permutation of L_1

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

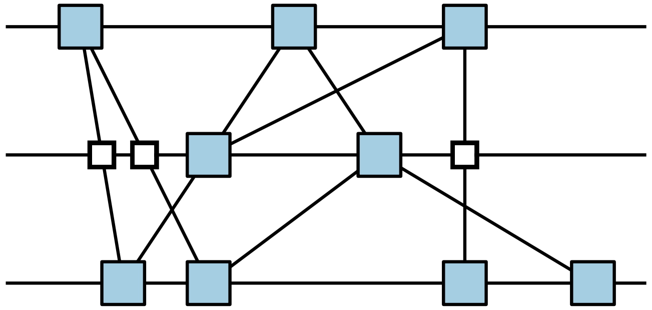
(3) minimize crossings by permuting L_{i+1} while keeping L_i fixed

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “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

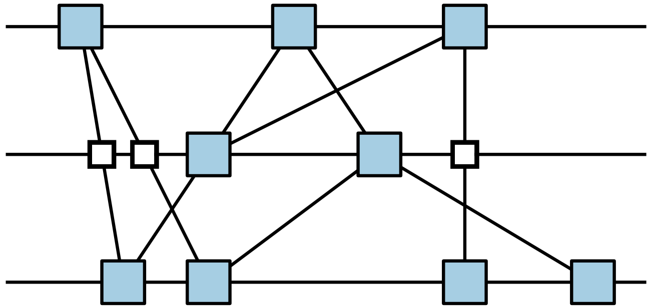
(4) repeat steps (2)–(3) in the reverse order (starting from topmost layer L_h)

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “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)

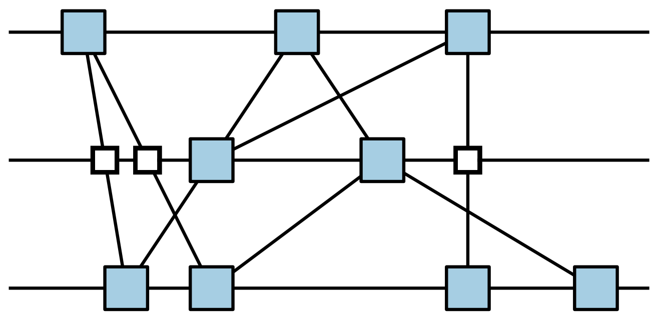
(5) repeat steps (2)–(4) until no further improvement is achieved

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “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

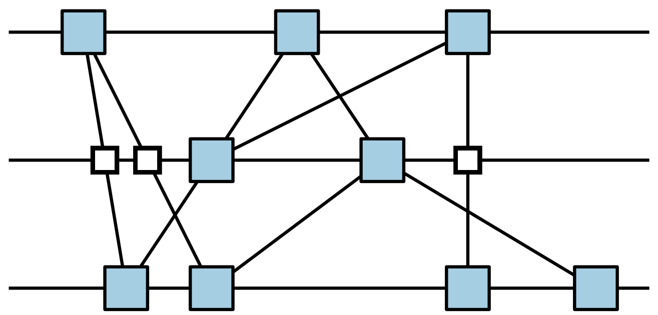
(6) repeat steps (1)–(5) with different starting permutations on L_1

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as "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

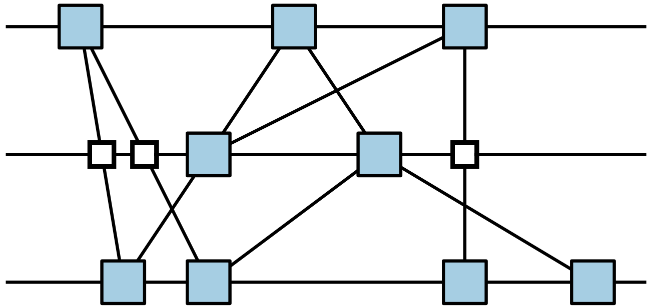
(6) repeat steps (1)–(5) with different starting permutations on L_1

Iterative Crossing Reduction

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

Idea.

- Permute one layer after the other.
- Treat dummy vertices as “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

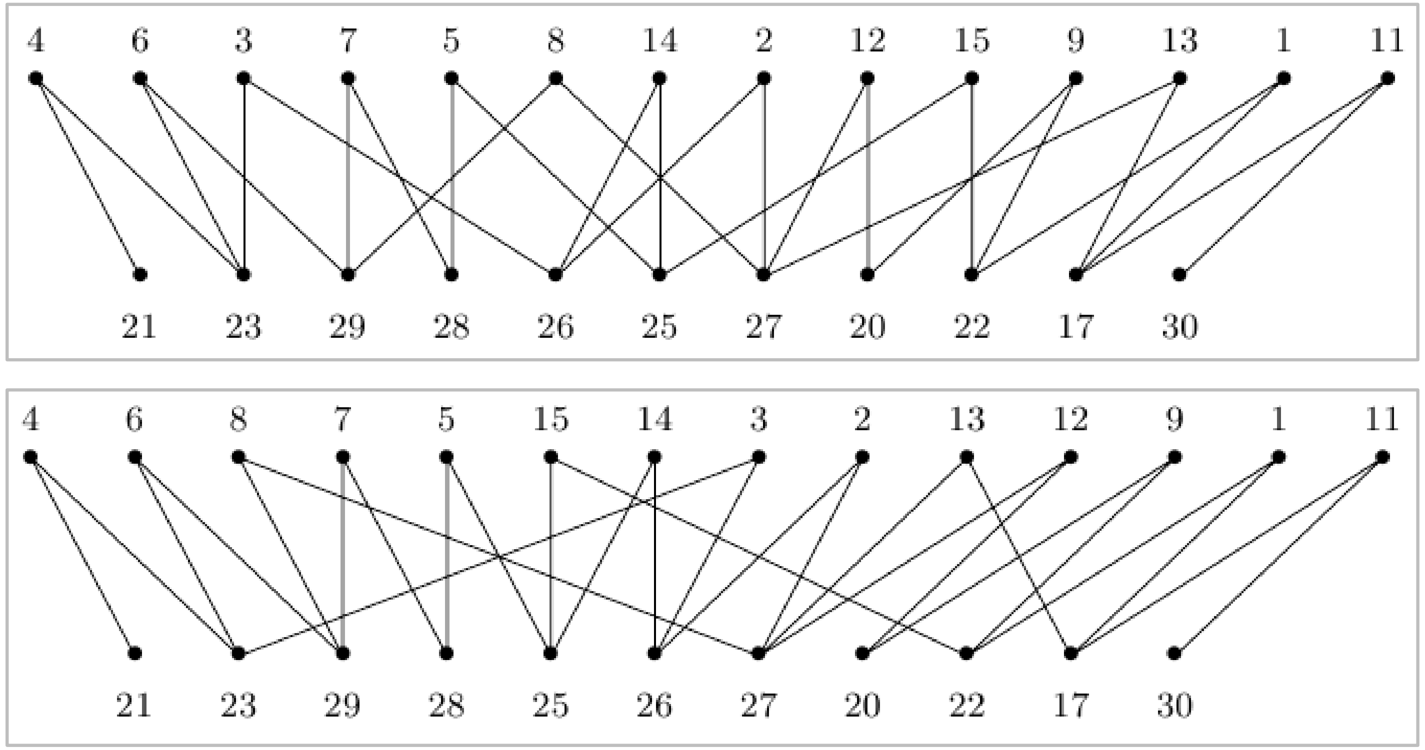
(4) repeat steps (2)–(3) in the reverse order (starting from topmost layer L_h)

(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.

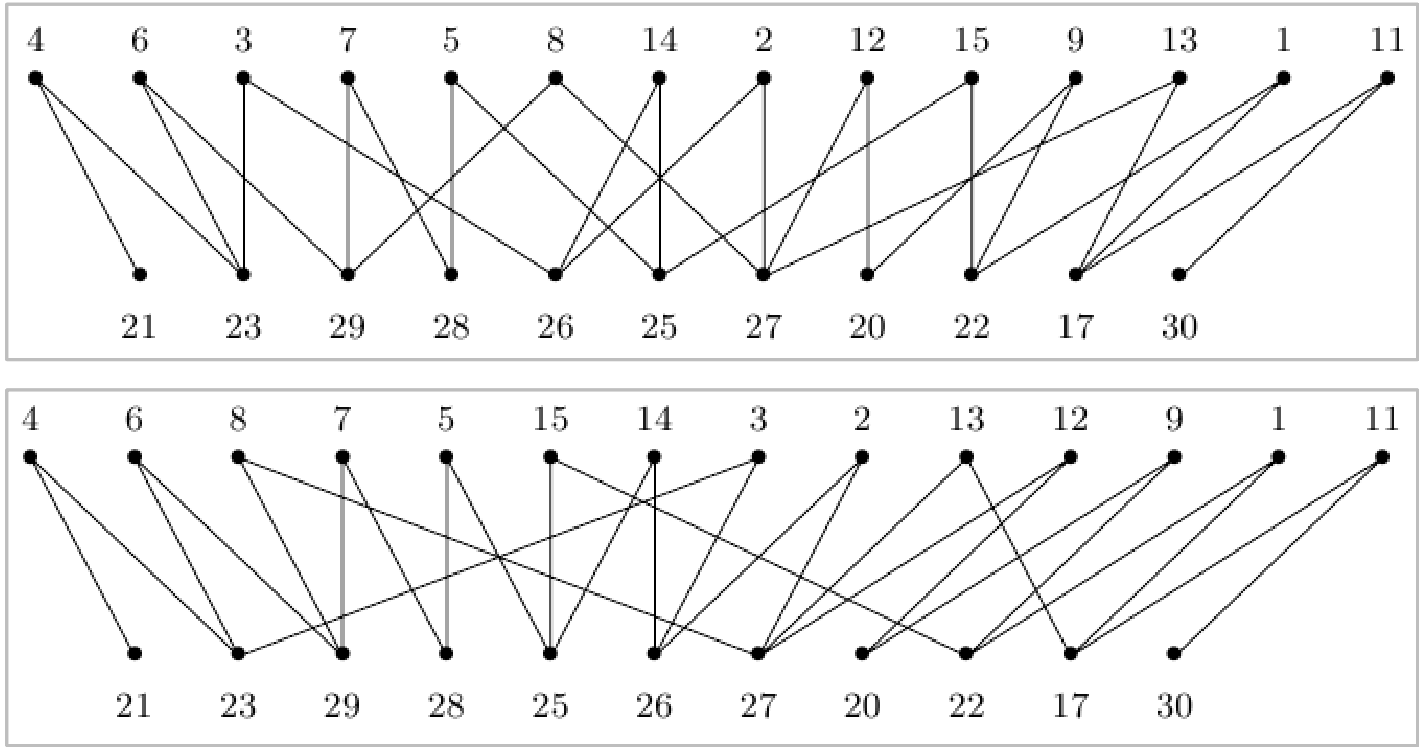


[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

Problem.

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

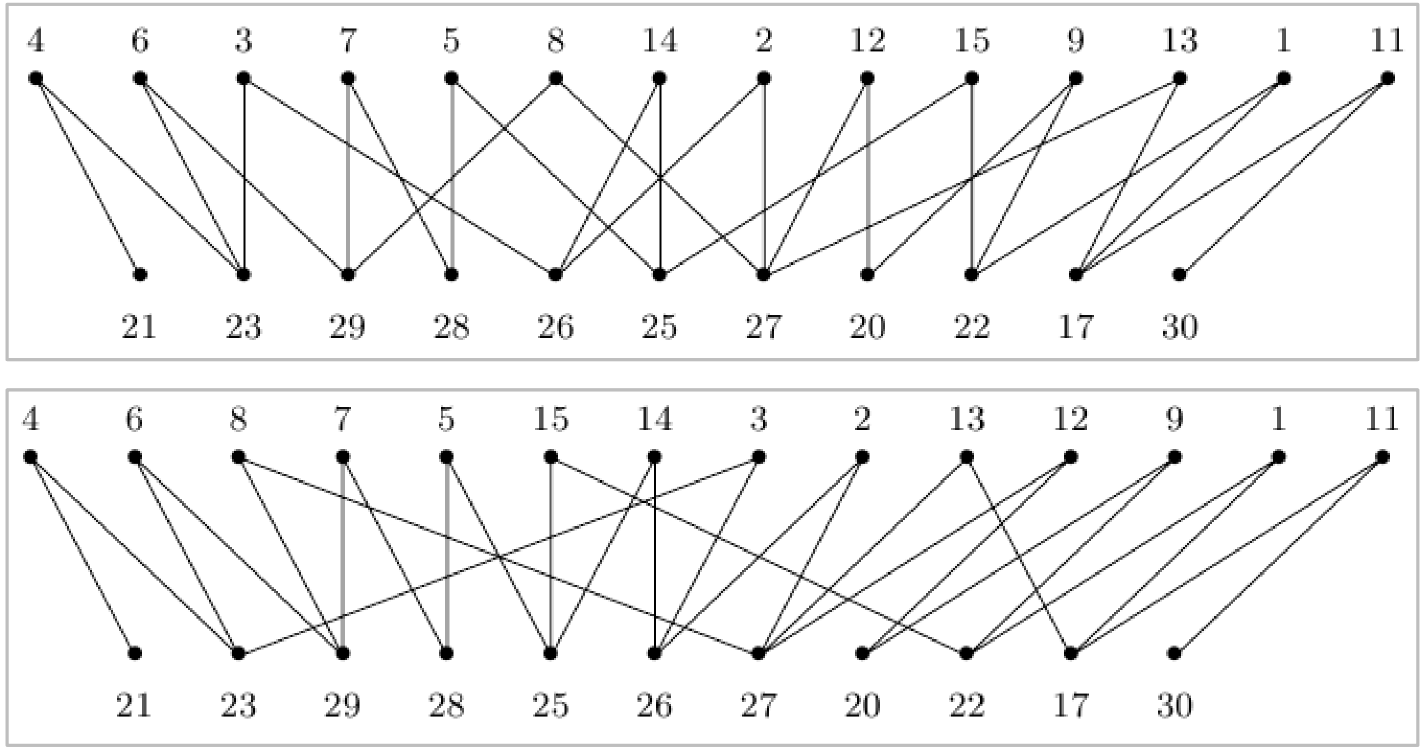


[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

Problem.

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



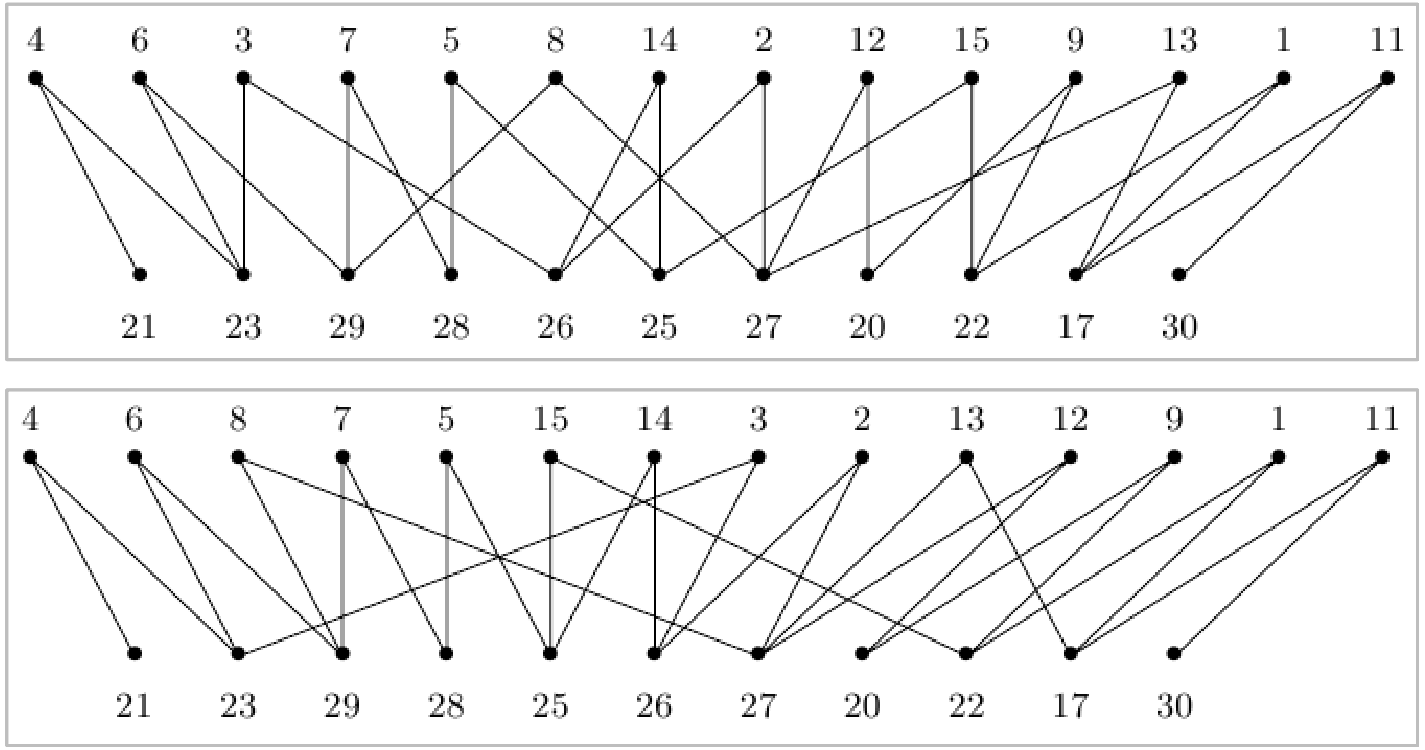
[Kaufmann & Wagner: Drawing Graphs]

One-Sided Crossing Minimization

Problem.

- Input: – bipartite graph G with $V(G) = L_1 \cup L_2$,
  – 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]

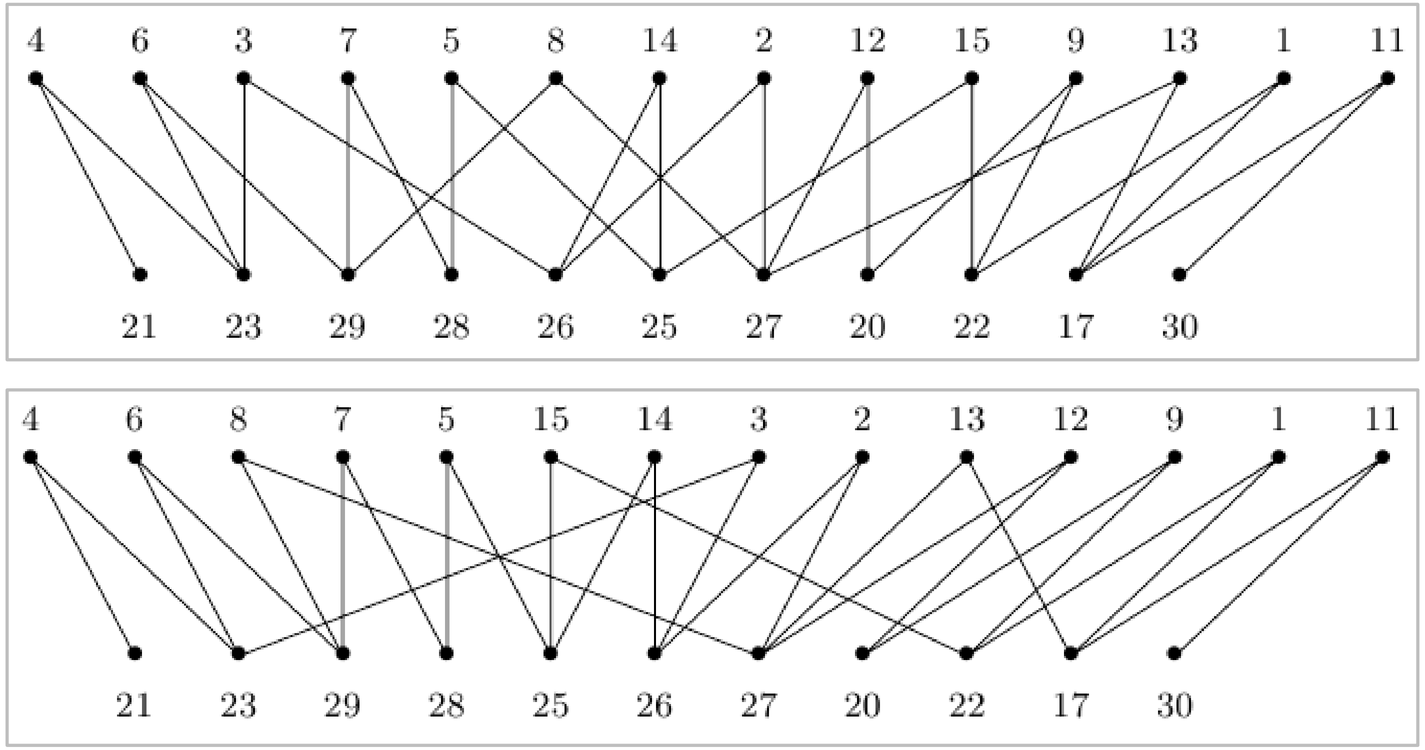
One-Sided Crossing Minimization

Problem.

- Input: – bipartite graph G with $V(G) = L_1 \cup L_2$,
  – 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

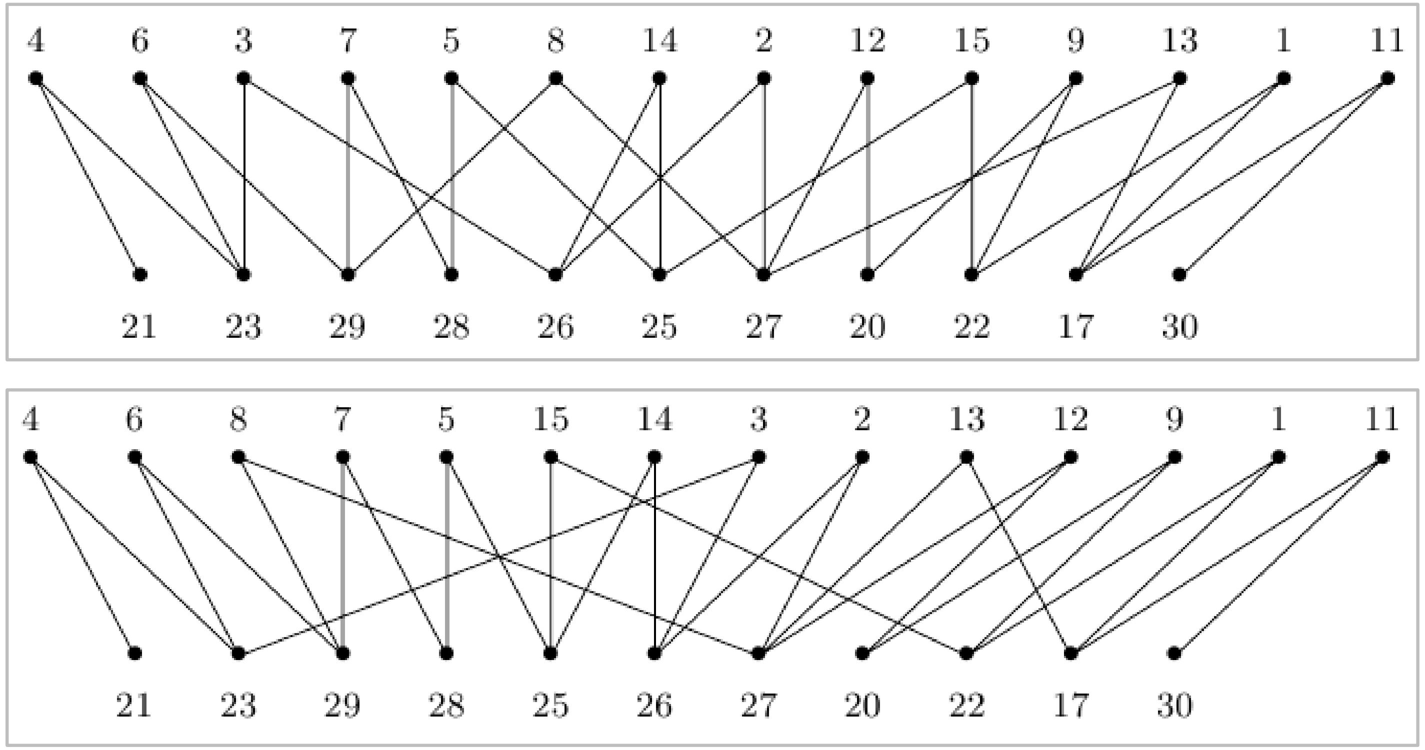
Problem.

- Input: – bipartite graph G with $V(G) = L_1 \cup L_2$,
  – 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

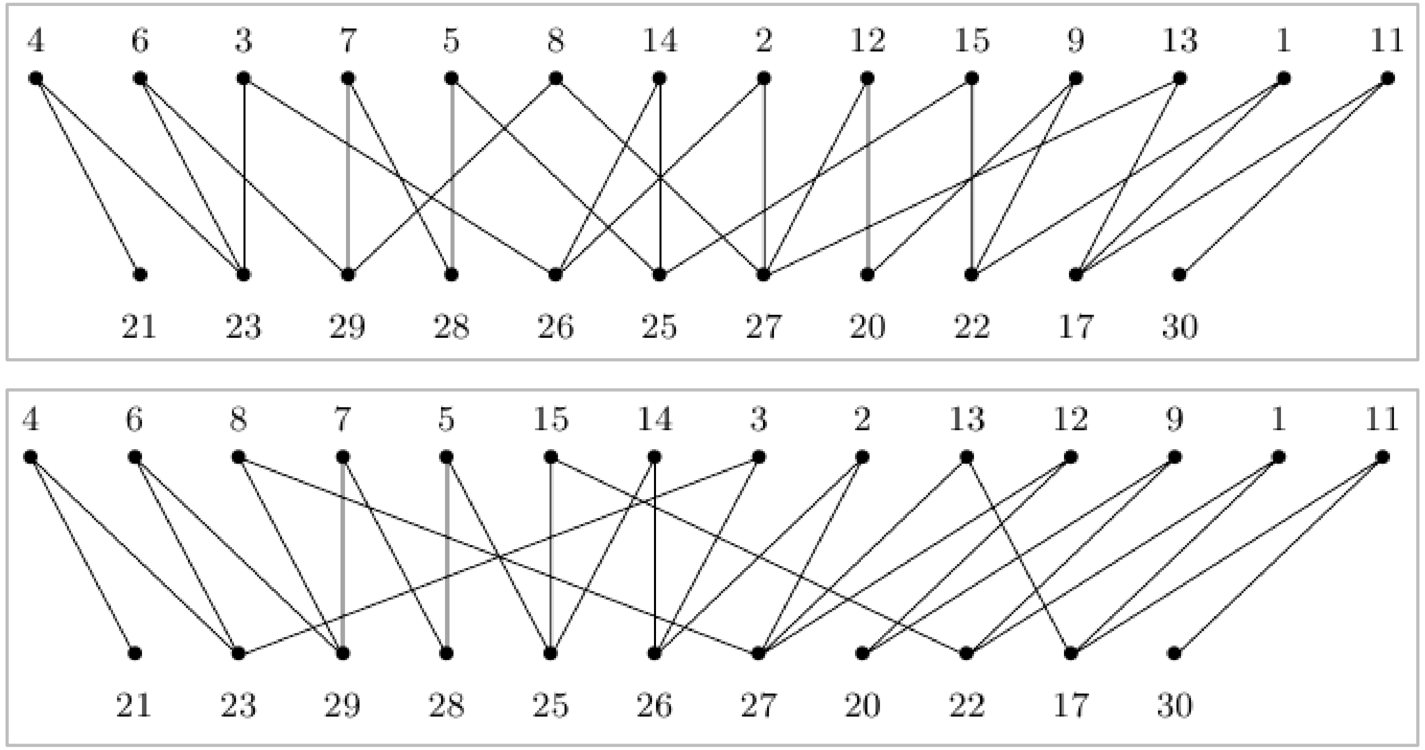
Problem.

- Input: – bipartite graph G with $V(G) = L_1 \cup L_2$,
  – 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

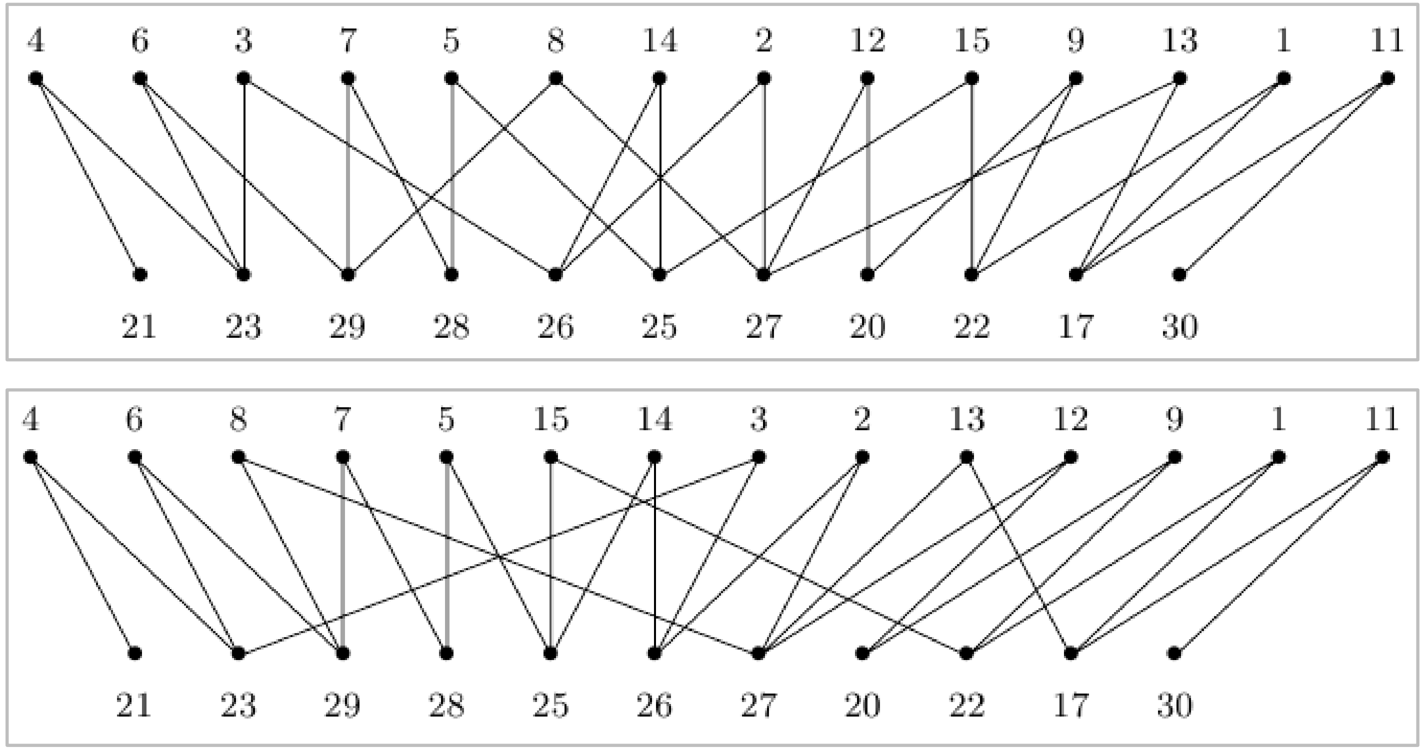
Problem.

- Input: – bipartite graph G with $V(G) = L_1 \cup L_2$,
  – 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

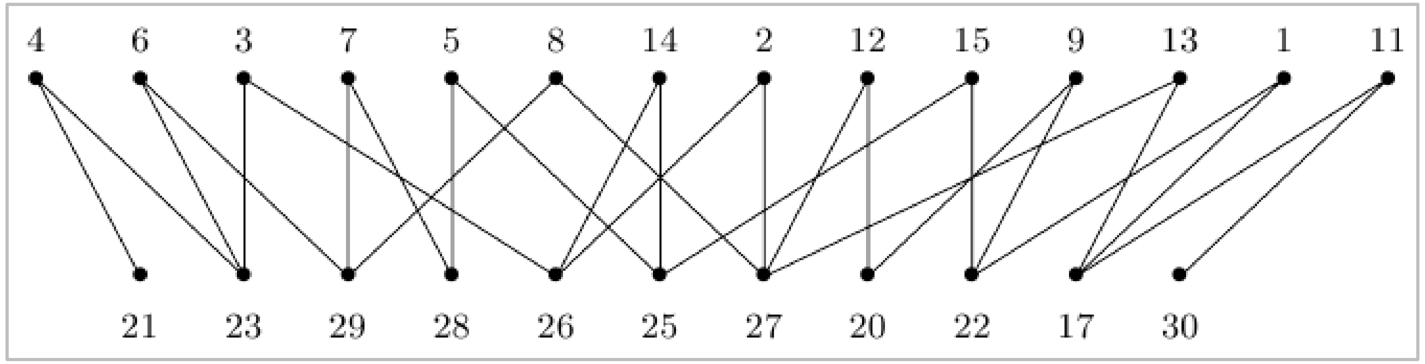
Problem.

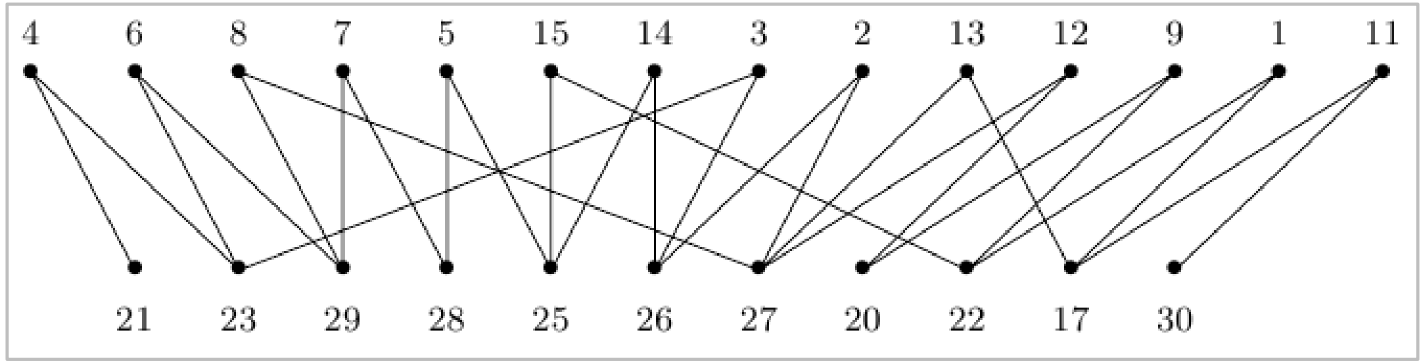
- Input: – bipartite graph G with $V(G) = L_1 \cup L_2$,
  – permutation π_1 on L_1
- Output: permutation π_2 of L_2 minimizing the number of edge crossings.

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

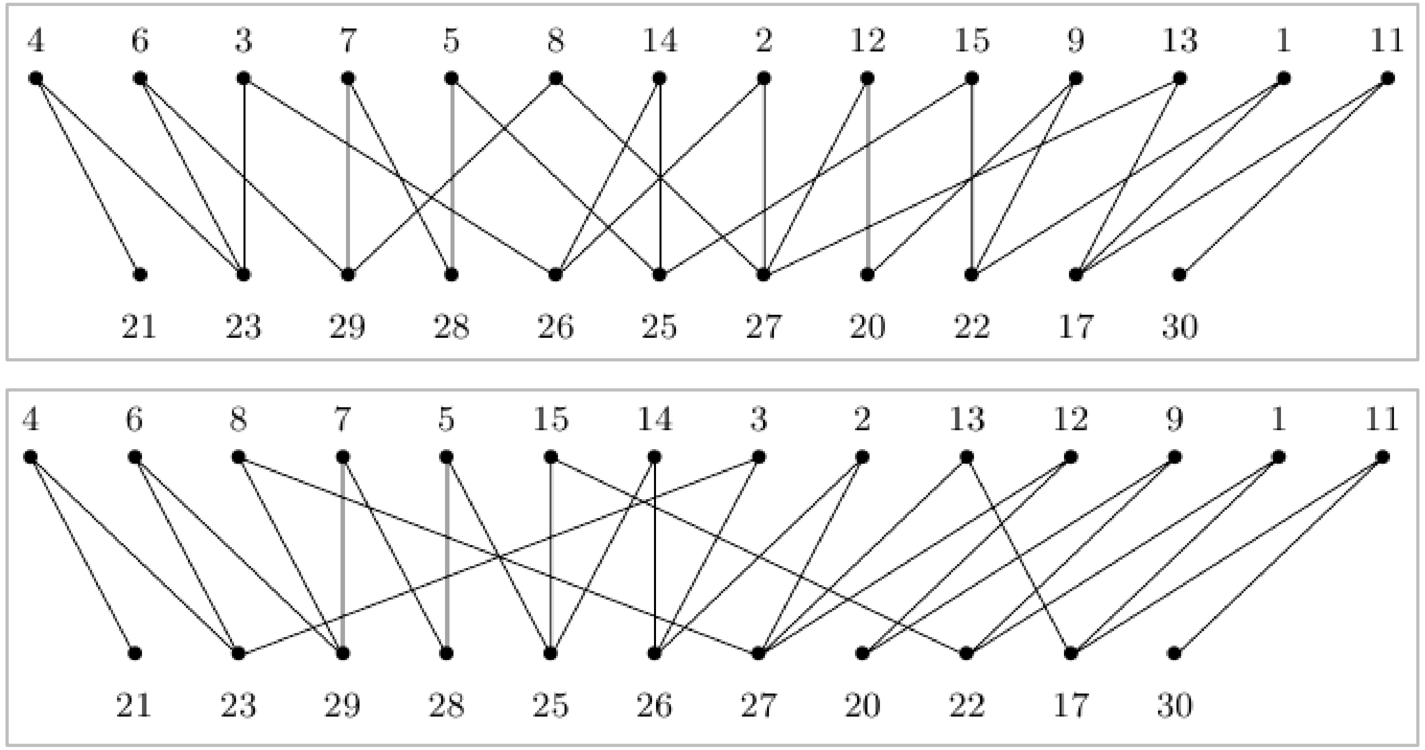
Problem.

- Input: – bipartite graph G with $V(G) = L_1 \cup L_2$,
  – 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: There are few crossing if vertices are "close" to their neighbors.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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: There are few crossing if 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:

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

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to $\mathrm{bary}(\cdot)$.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to $\mathrm{bary}(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to $\mathrm{bary}(\cdot)$.
- 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: There are few crossing if 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 according to bary$(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to $\mathrm{bary}(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to $\mathrm{bary}(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist.
- Factor-$O(\sqrt{|V(G)|})$ approximation.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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:

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

- To get π_2, sort L_2 ascendingly according to $\mathrm{bary}(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ⟵ Exercise!
- Factor-$O(\sqrt{|V(G)|})$ approximation.

Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to $\text{bary}(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-$O(\sqrt{|V(G)|})$ approximation.

Barycenter Heuristic

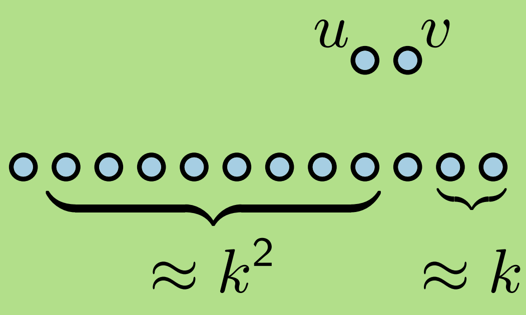
[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to $\text{bary}(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-$O(\sqrt{|V(G)|})$ approximation.

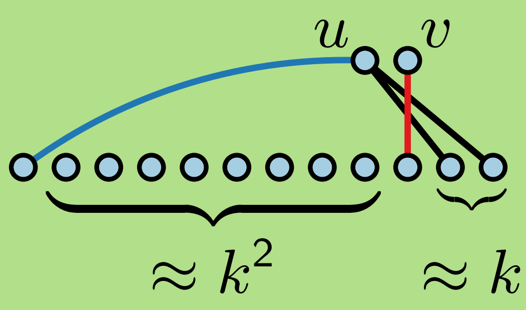
Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if 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 according to bary$(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-$O(\sqrt{|V(G)|})$ approximation.

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 $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $\text{med}(u) :=$

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

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

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\mathrm{med}(u) := \begin{cases} 0 & \text{if } N(u) = \emptyset, \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise.} \end{cases}$$
- To get π_2, sort L_2 ascendingly according to $\mathrm{med}(\cdot)$.
- For vertices with the same median, place vertices of odd degree to the left of vertices of even degree (and keep the old relative ranks among the odd/even-degree vertices).
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist.

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{if } N(u) = \emptyset, \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise.} \end{cases}$$
- To get π_2, sort L_2 ascendingly according to med$(\cdot)$.
- For vertices with the same median, place vertices of odd degree to the left of vertices of even degree (and keep the old relative ranks among the odd/even-degree vertices).
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \dots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \dots < \pi_1(v_k)$
- $$\mathrm{med}(u) := \begin{cases} 0 & \text{if } N(u) = \emptyset, \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise.} \end{cases}$$
- To get π_2, sort L_2 ascendingly according to $\mathrm{med}(\cdot)$.
- For vertices with the same median, place vertices of odd degree to the left of vertices of even degree (and keep the old relative ranks among the odd/even-degree vertices).
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-3 approximation.

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{if } N(u) = \emptyset, \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise.} \end{cases}$$
- To get π_2, sort L_2 ascendingly according to $\text{med}(\cdot)$.
- For vertices with the same median, place vertices of odd degree to the left of vertices of even degree (and keep the old relative ranks among the odd/even-degree vertices).
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-3 approximation. Proof in [GD Ch 11]

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{if } N(u) = \emptyset, \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise.} \end{cases}$$
- To get π_2, sort L_2 ascendingly according to med$(\cdot)$.
- For vertices with the same median, place vertices of odd degree to the left of vertices of even degree (and keep the old relative ranks among the odd/even-degree vertices).
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-3 approximation. Proof in [GD Ch 11]

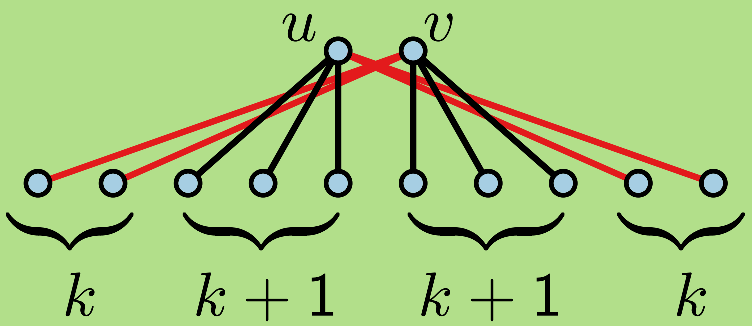
Worst case?

Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\mathrm{med}(u) := \begin{cases} 0 & \text{if } N(u) = \emptyset, \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise.} \end{cases}$$
- To get π_2, sort L_2 ascendingly according to $\mathrm{med}(\cdot)$.
- For vertices with the same median, place vertices of odd degree to the left of vertices of even degree (and keep the old relative ranks among the odd/even-degree vertices).
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-3 approximation. Proof in [GD Ch 11]

Worst case?

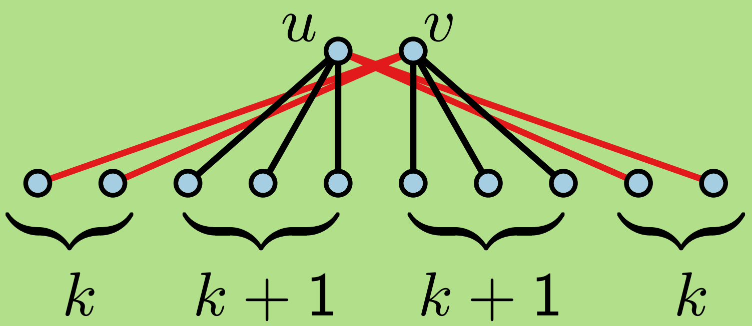


Median Heuristic

[Eades & Wormald '94]

- $\{v_1, \ldots, v_k\} := N(u)$ with $\pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k)$
- $$\text{med}(u) := \begin{cases} 0 & \text{if } N(u) = \emptyset, \\ \pi_1(v_{\lceil k/2 \rceil}) & \text{otherwise.} \end{cases}$$
- To get π_2, sort L_2 ascendingly according to med$(\cdot)$.
- For vertices with the same median, place vertices of odd degree to the left of vertices of even degree (and keep the old relative ranks among the odd/even-degree vertices).
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-3 approximation. 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 $\Rightarrow O(|L_2|^2)$ time.

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 $\Rightarrow O(|L_2|^2)$ time.
- 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 $\Rightarrow O(|L_2|^2)$ time.
- Suitable as post-processing for other heuristics.

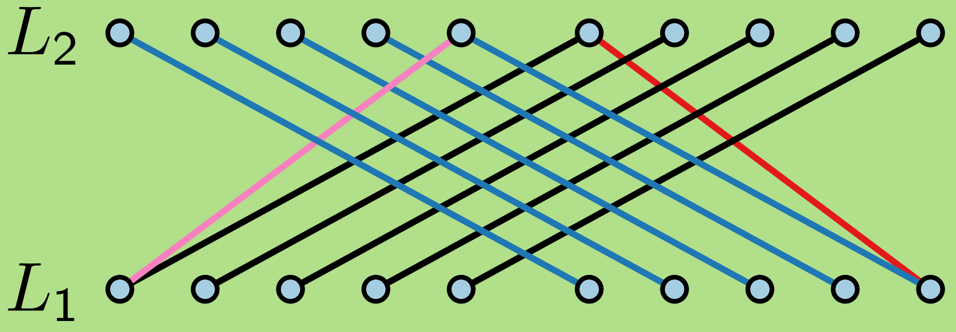
Worst case?

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 $\Rightarrow O(|L_2|^2)$ time.
- Suitable as post-processing for other heuristics.

Worst case?

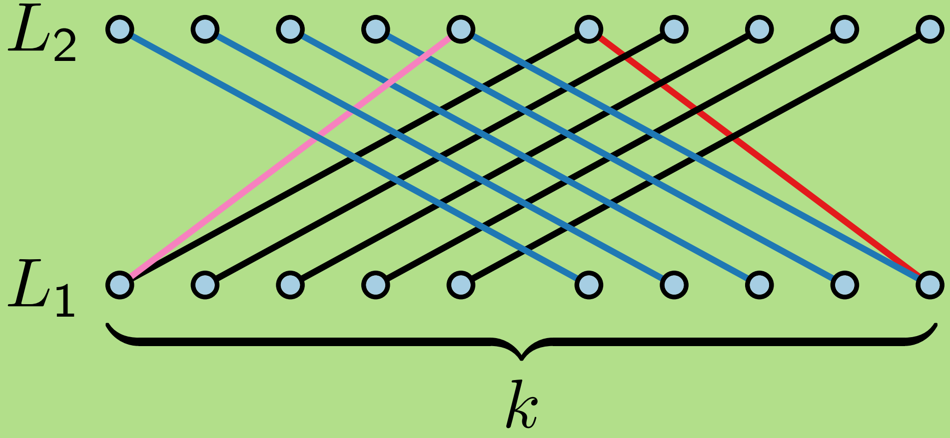


Greedy-Switch Heuristic

[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 $\Rightarrow O(|L_2|^2)$ time.
- Suitable as post-processing for other heuristics.

Worst case?

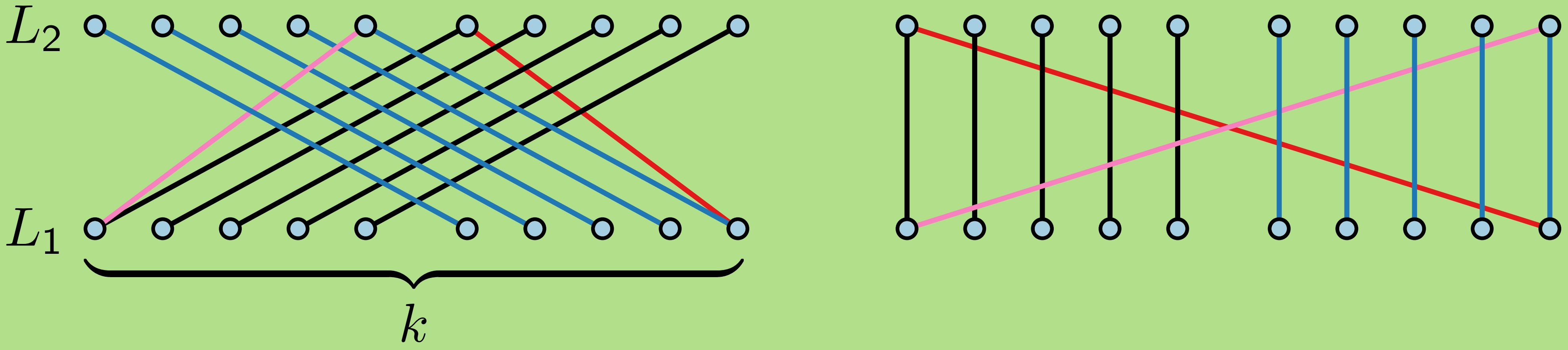


Greedy-Switch Heuristic

[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 $\Rightarrow O(|L_2|^2)$ time.
- Suitable as post-processing for other heuristics.

Worst case?

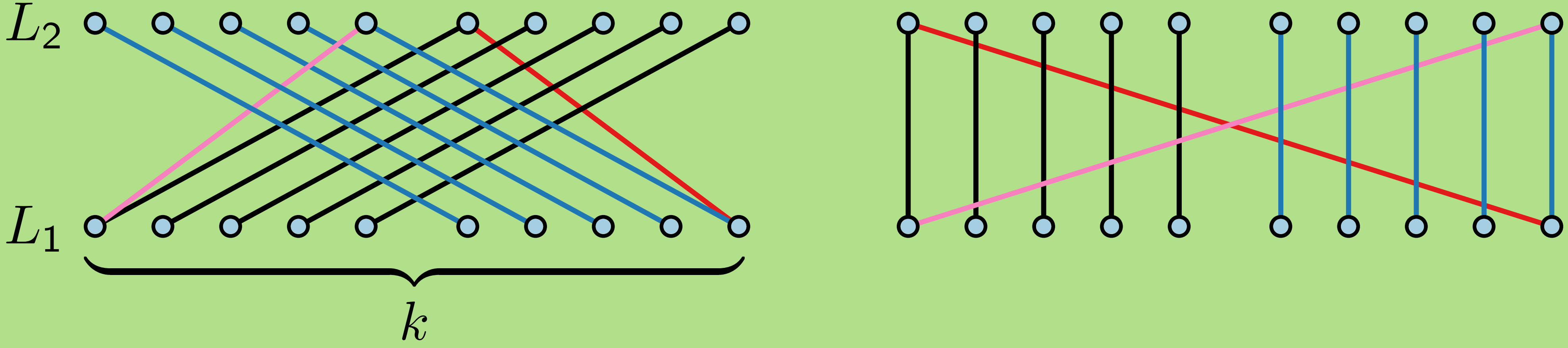


Greedy-Switch Heuristic

[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 $\Rightarrow O(|L_2|^2)$ time.
- Suitable as post-processing for other heuristics.

Worst case?

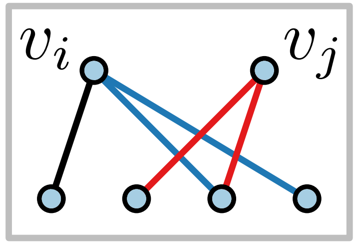


crossings: $\approx k^2/4$ $\approx 2k$

Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

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

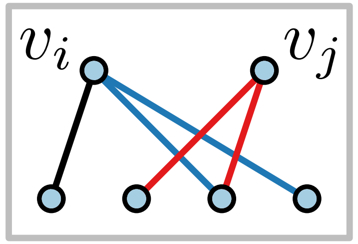


$c_{ij} = 3$

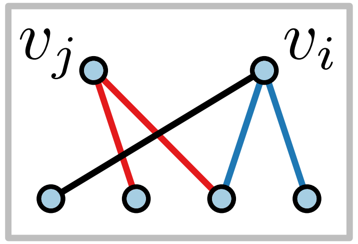
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

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



$c_{ij} = 3$



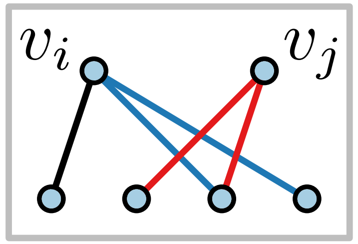
$c_{ji} = 2$

Integer Linear Program (ILP)

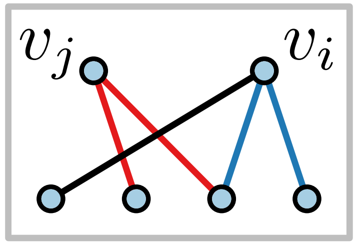
[Jünger & Mutzel, '97]

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

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



$c_{ij} = 3$



$c_{ji} = 2$

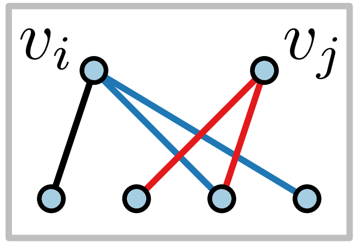
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

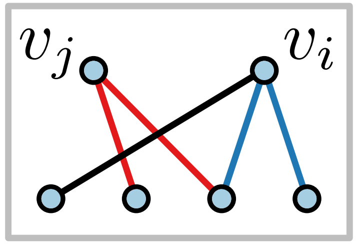
- Constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j if $\pi_2(v_i) < \pi_2(v_j)$

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

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



$c_{ij} = 3$



$c_{ji} = 2$

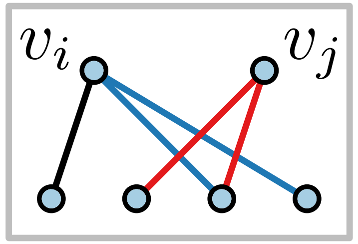
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

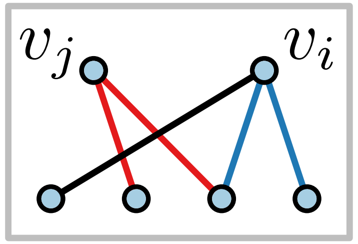
- Constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j if $\pi_2(v_i) < \pi_2(v_j)$

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

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



$c_{ij} = 3$



$c_{ji} = 2$

- Number of crossings of a permutations π_2:

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

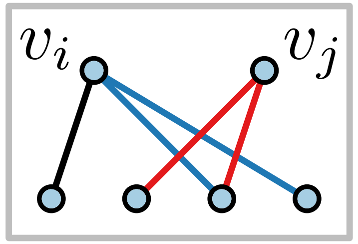
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

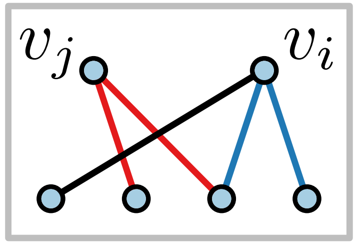
- Constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j if $\pi_2(v_i) < \pi_2(v_j)$

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

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



$c_{ij} = 3$



$c_{ji} = 2$

- Number of crossings of a permutations π_2:

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

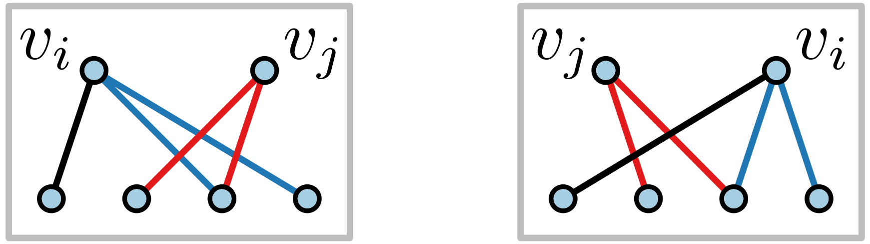
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

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

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

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



$c_{ij} = 3$ $c_{ji} = 2$

- Number of crossings of a permutations π_2:

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

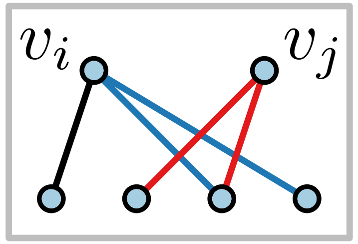
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

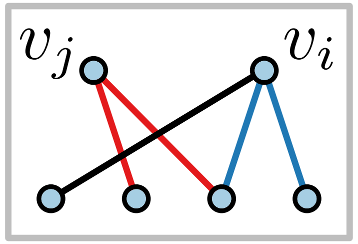
- Constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j if $\pi_2(v_i) < \pi_2(v_j)$

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

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



$c_{ij} = 3$



$c_{ji} = 2$

- Number of crossings of a permutations π_2:

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

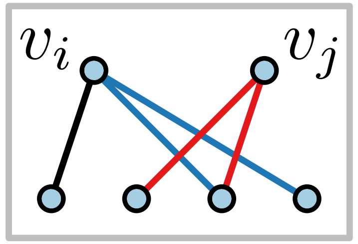
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

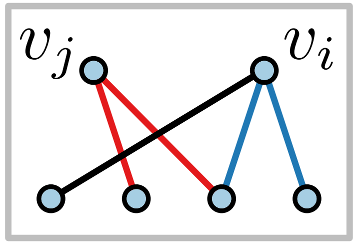
- Constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j if $\pi_2(v_i) < \pi_2(v_j)$

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

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



$c_{ij} = 3$



$c_{ji} = 2$

- Number of crossings of a permutations π_2:

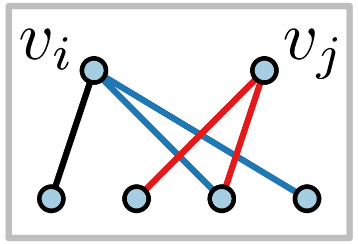
$$\mathrm{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji}) x_{ij} + \underbrace{\sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji}}_{\text{constant}}$$

Integer Linear Program (ILP)

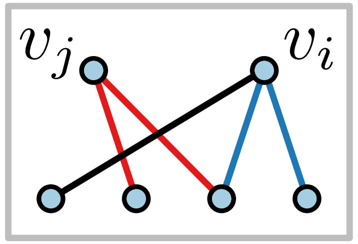
[Jünger & Mutzel, '97]

- Constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j if $\pi_2(v_i) < \pi_2(v_j)$
- Variable x_{ij} for each $1 \leq i < j \leq n_2 := |L_2|$

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



$c_{ij} = 3$



$c_{ji} = 2$

- Number of crossings of a permutations π_2:

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

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- Transitivity constraints:

$$\text{for } 1 \leq i < j < k \leq n_2$$

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- Transitivity constraints:

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)

- Objective (minimize the number of crossings):

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

- Transitivity constraints:

for $1 \leq i < j < k \leq n_2$

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

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- 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 $x_{ij} = \substack{1 \\ 0}$ and $x_{jk} = \substack{1 \\ 0}$, then $x_{ik} = \substack{1 \\ 0}$

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- 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 $x_{ij} = \underset{0}{1}$ and $x_{jk} = \underset{0}{1}$, then $x_{ik} = \underset{0}{1}$

Properties.

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- 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 $x_{ij} = \begin{matrix}1\\0\end{matrix}$ and $x_{jk} = \begin{matrix}1\\0\end{matrix}$, then $x_{ik} = \begin{matrix}1\\0\end{matrix}$

Properties.

- branch-and-cut technique applicable for this ILP

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- 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 $x_{ij} = \genfrac{}{}{0pt}{}{1}{0}$ and $x_{jk} = \genfrac{}{}{0pt}{}{1}{0}$, then $x_{ik} = \genfrac{}{}{0pt}{}{1}{0}$

Properties.

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

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- 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 $x_{ij} = \overset{1}{0}$ and $x_{jk} = \overset{1}{0}$, then $x_{ik} = \overset{1}{0}$

Properties.

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

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- Transitivity constraints:

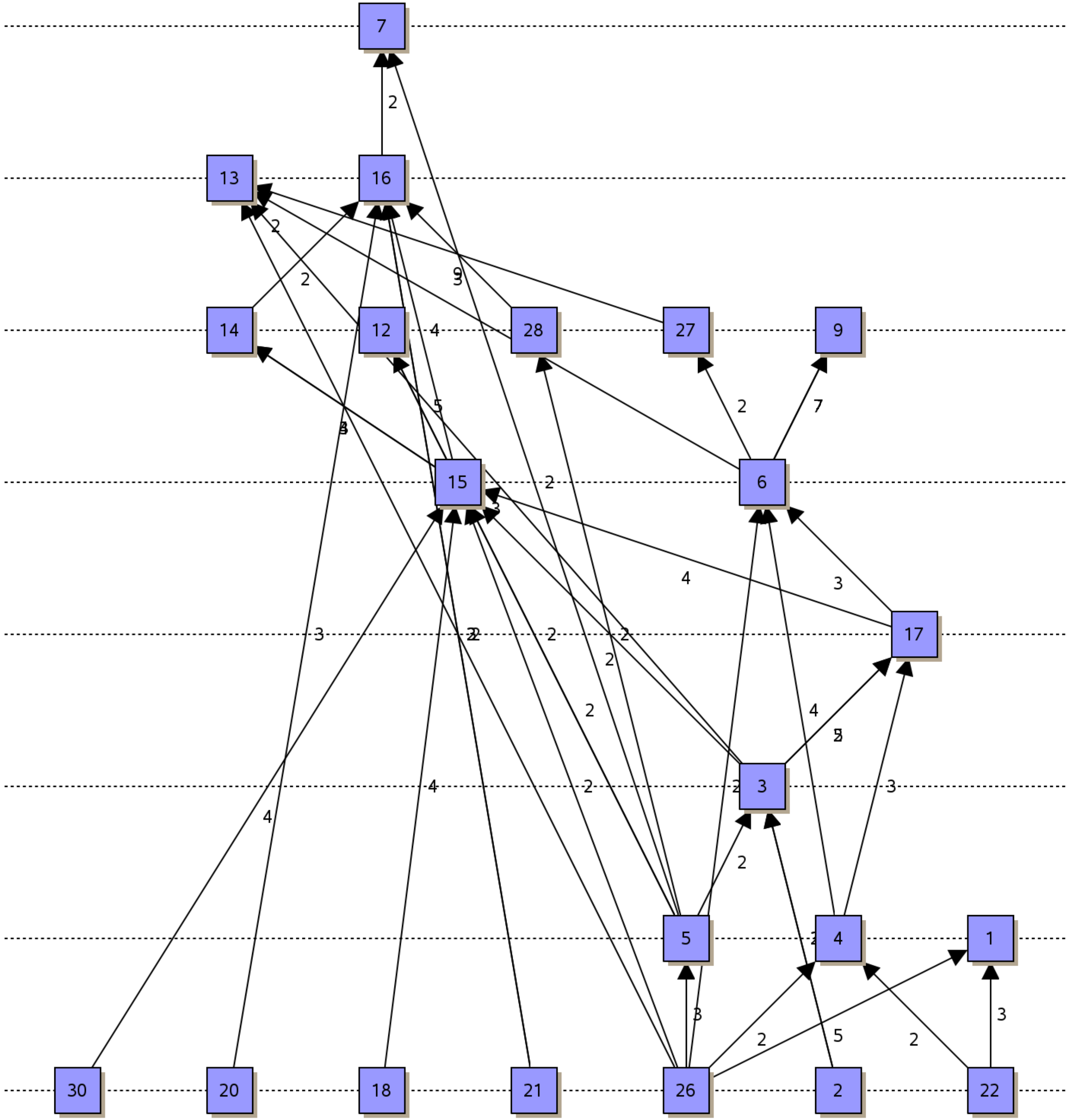
$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

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

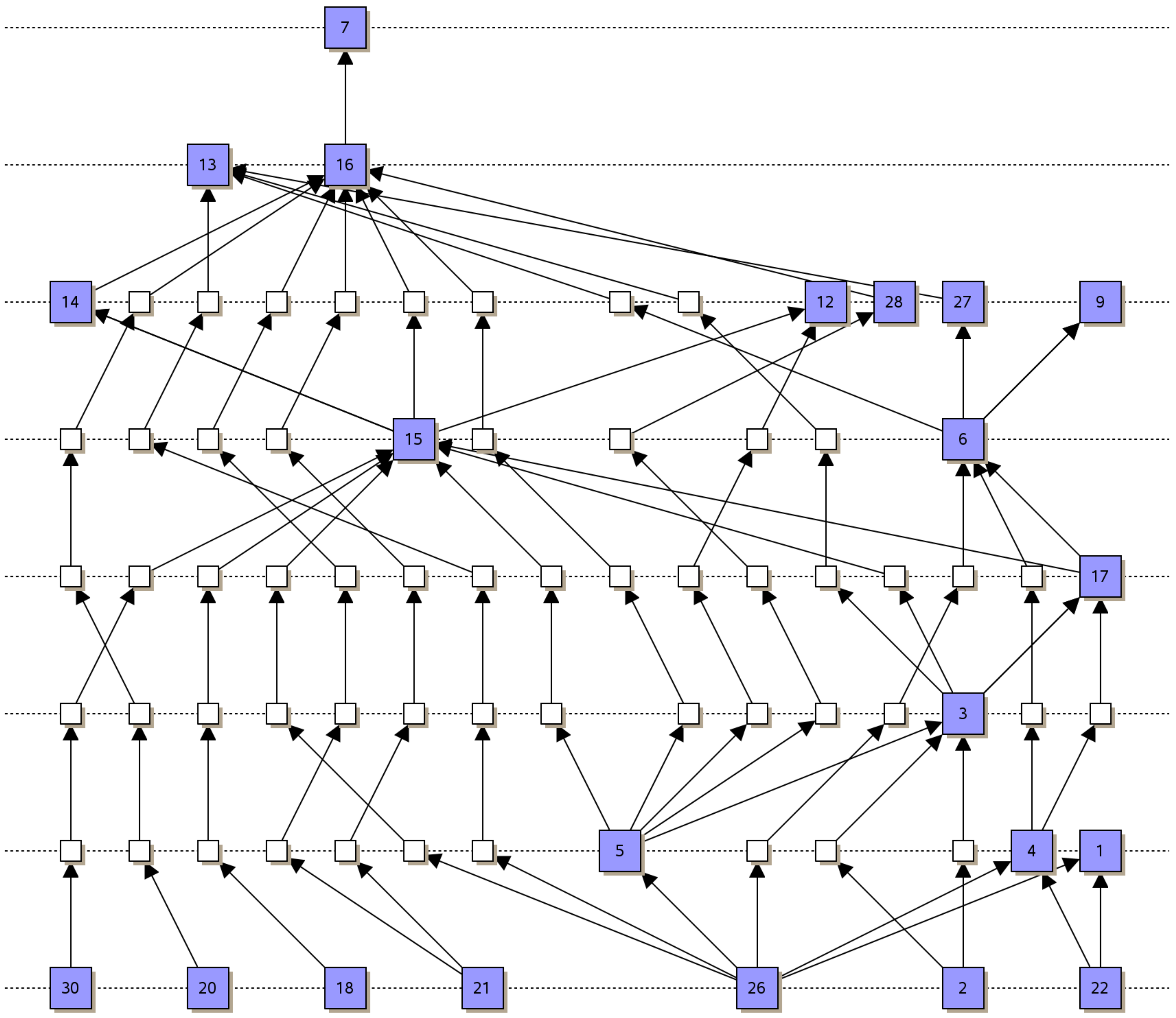
Properties.

- 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

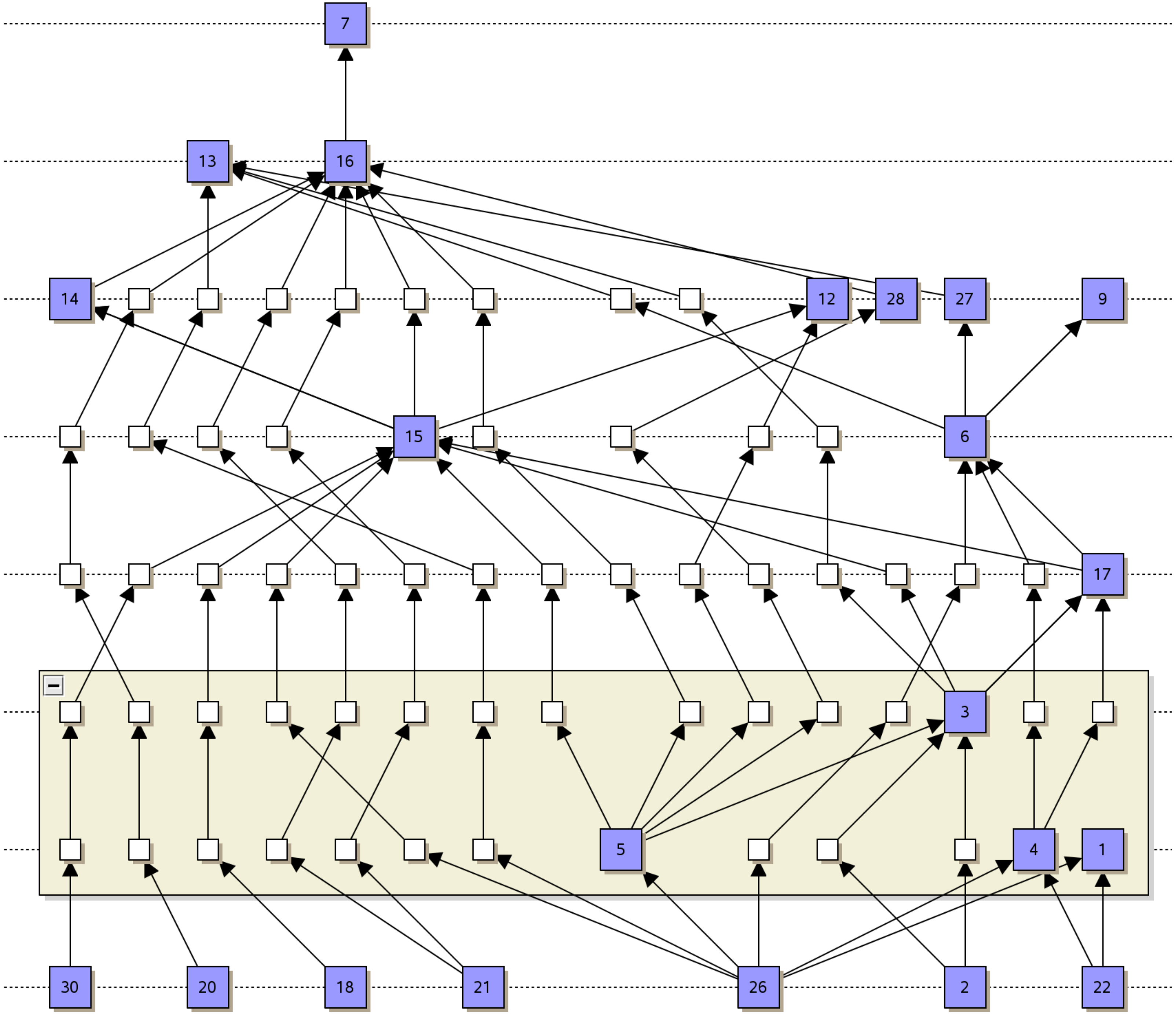
Iterations on Example



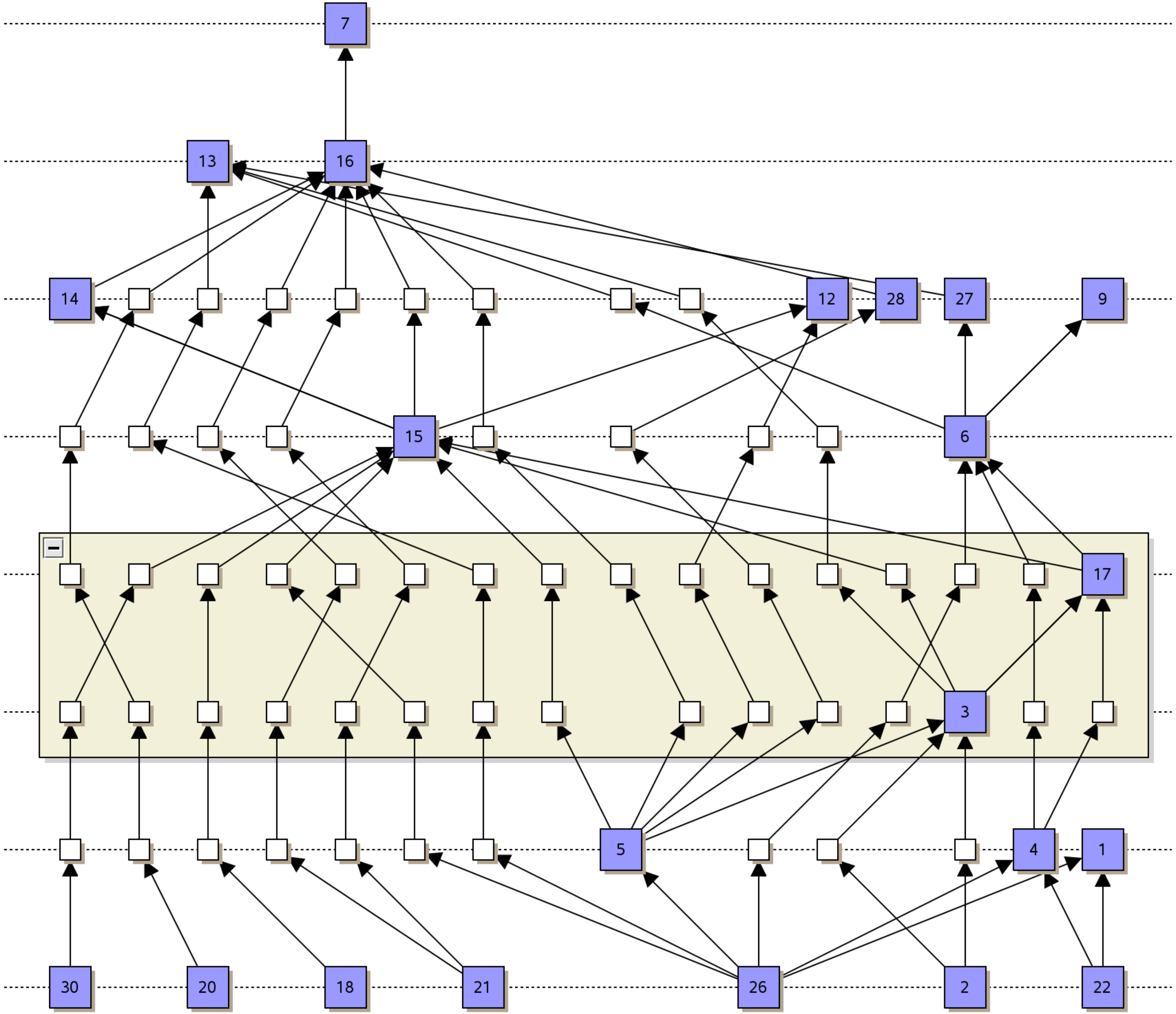
Iterations on Example



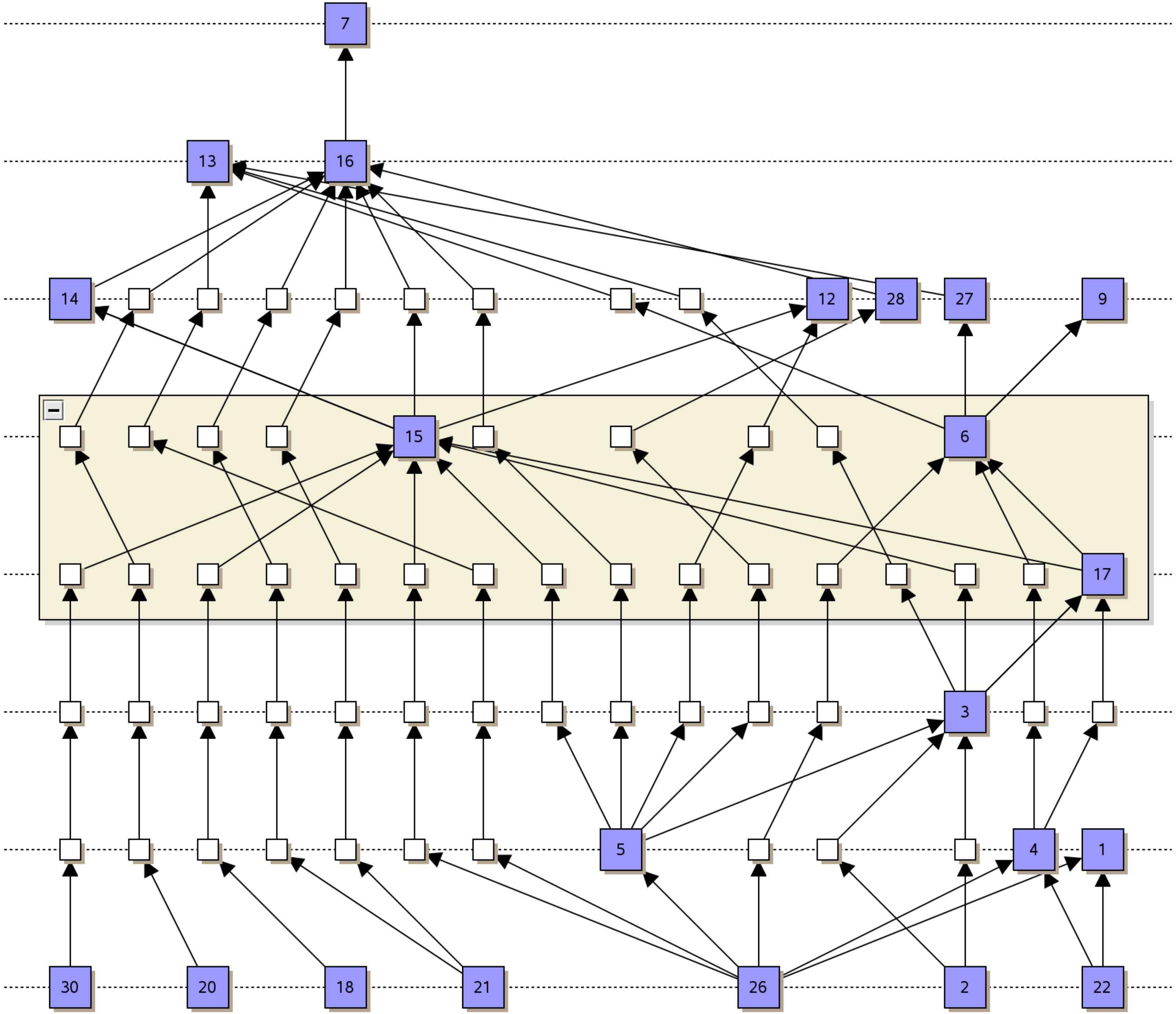
Iterations on Example



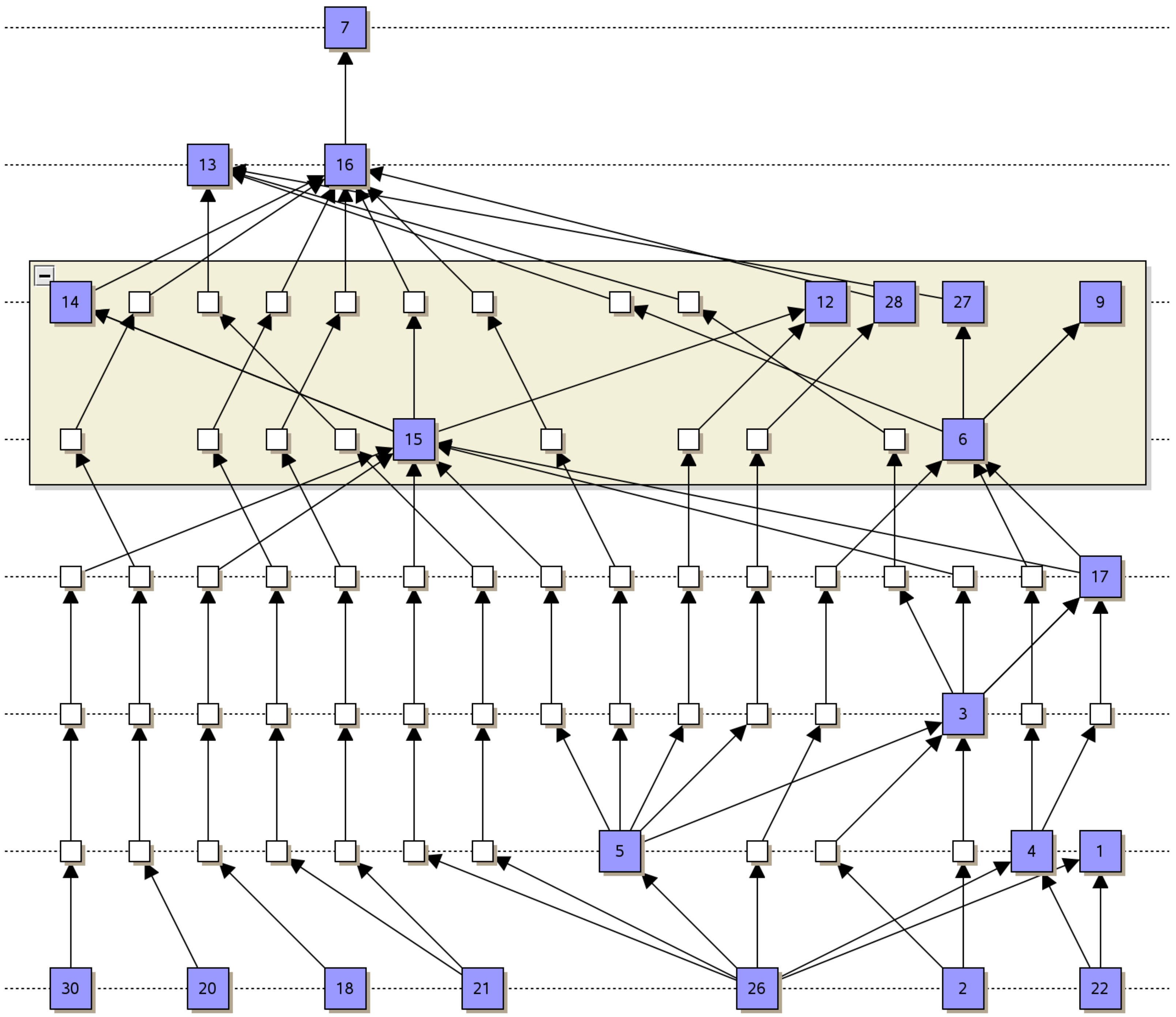
Iterations on Example



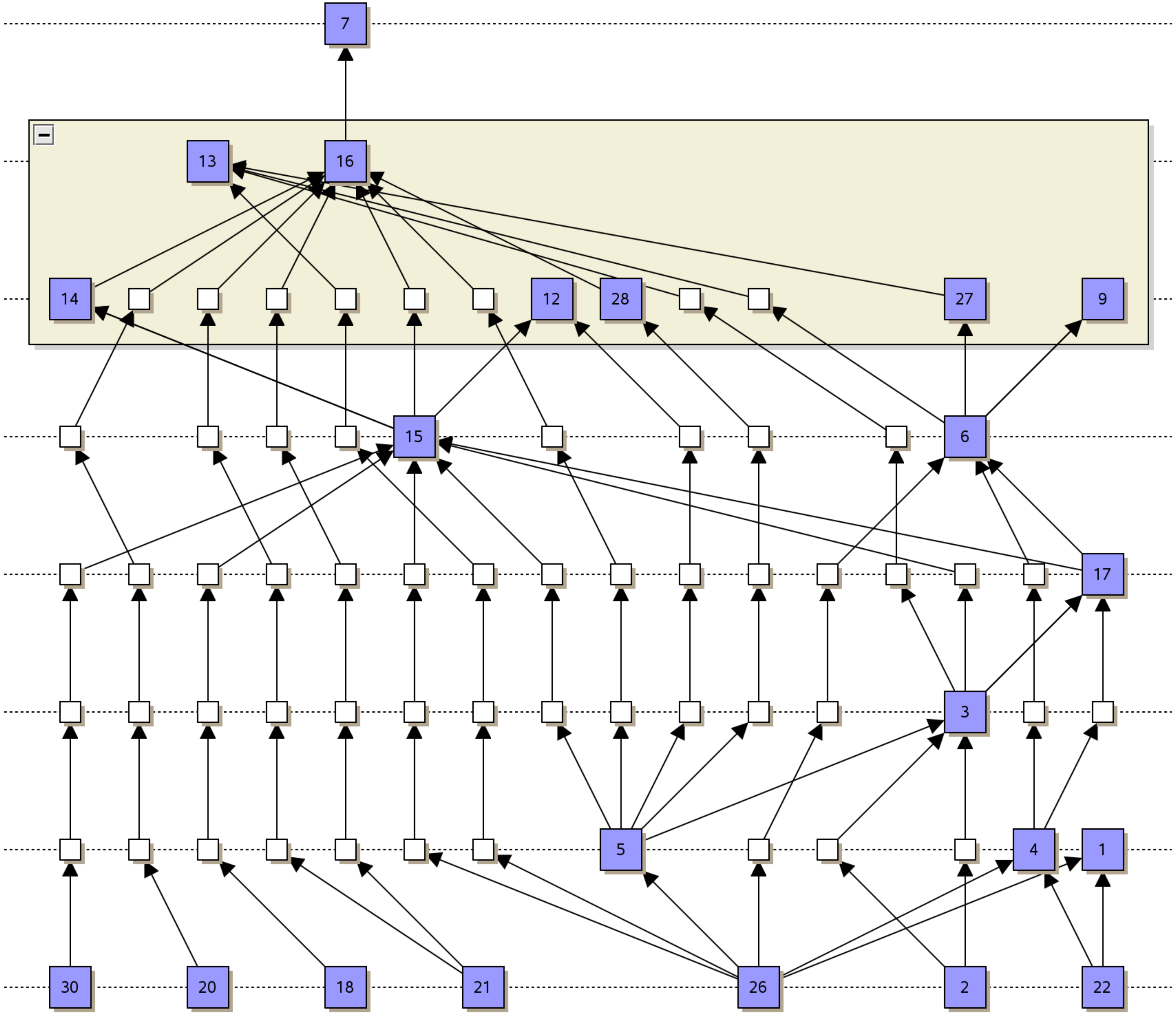
Iterations on Example



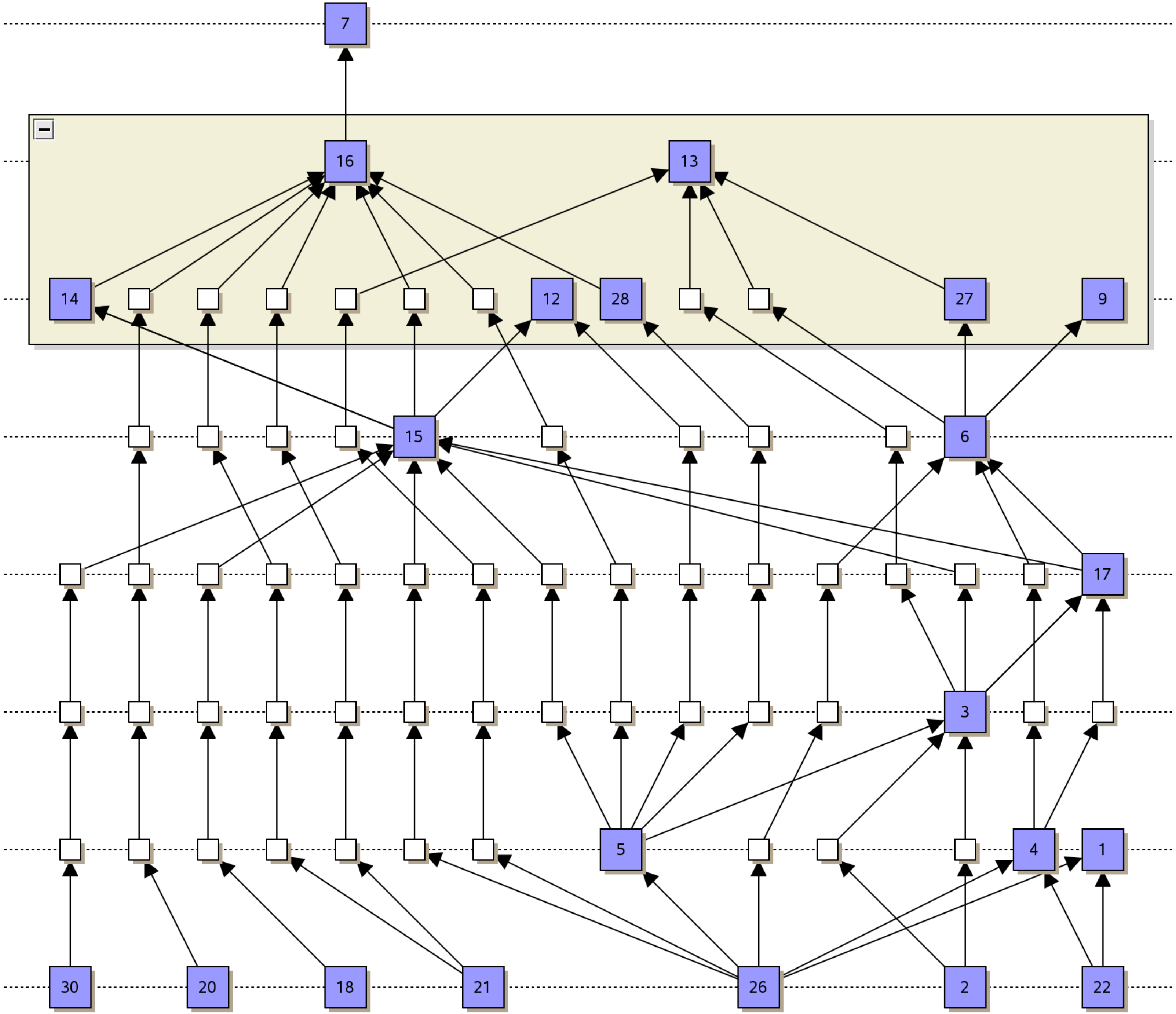
Iterations on Example



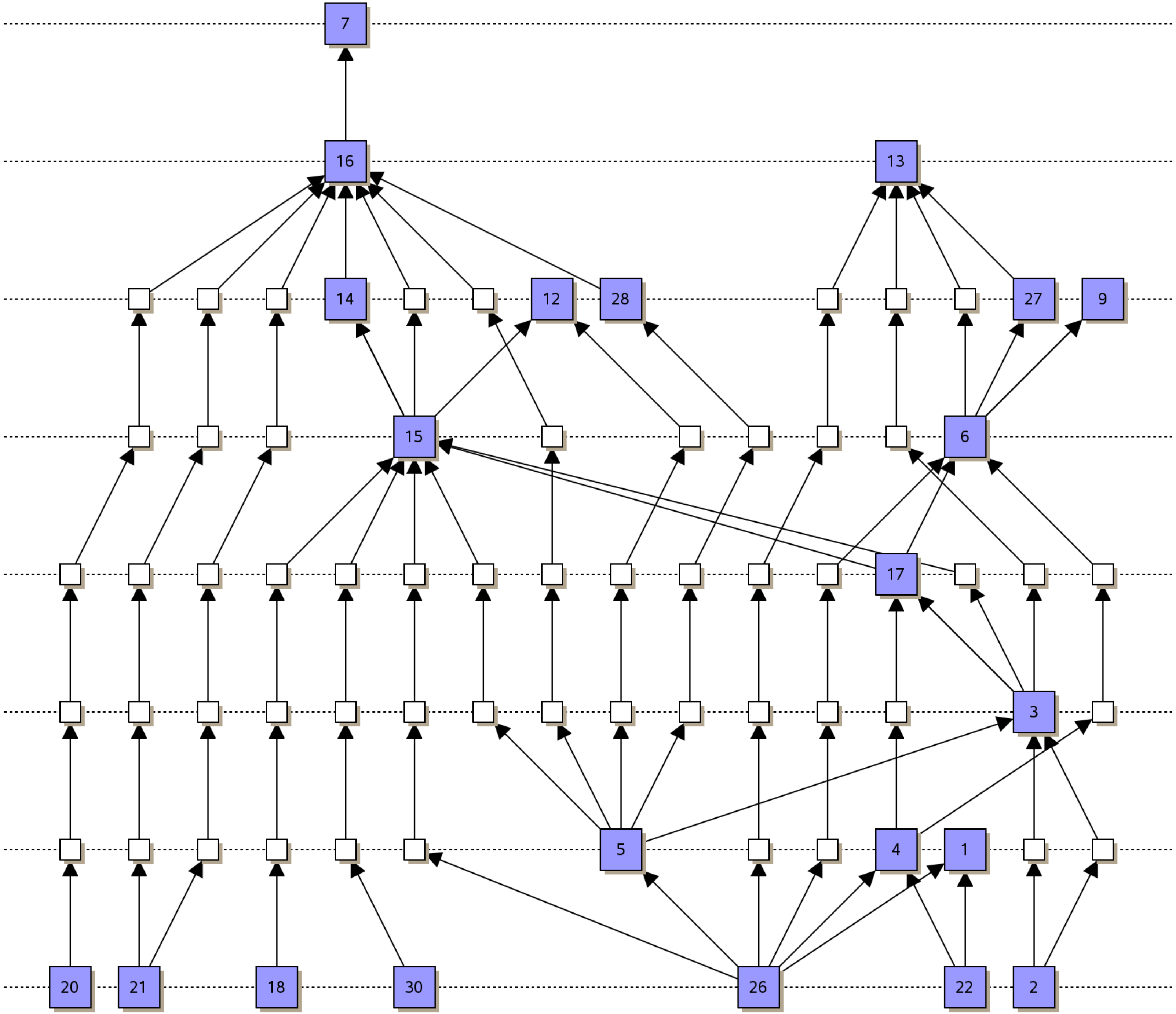
Iterations on Example



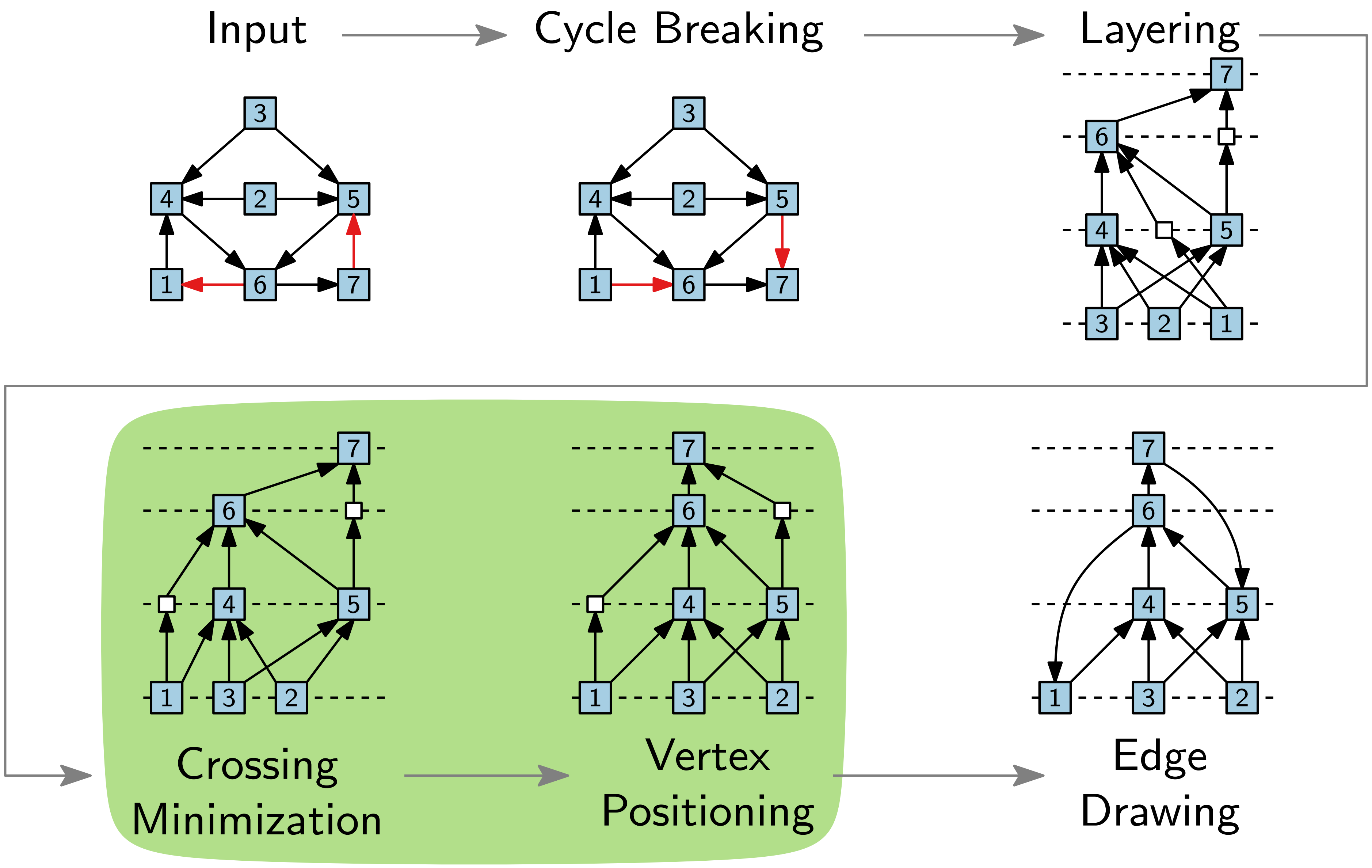
Iterations on Example



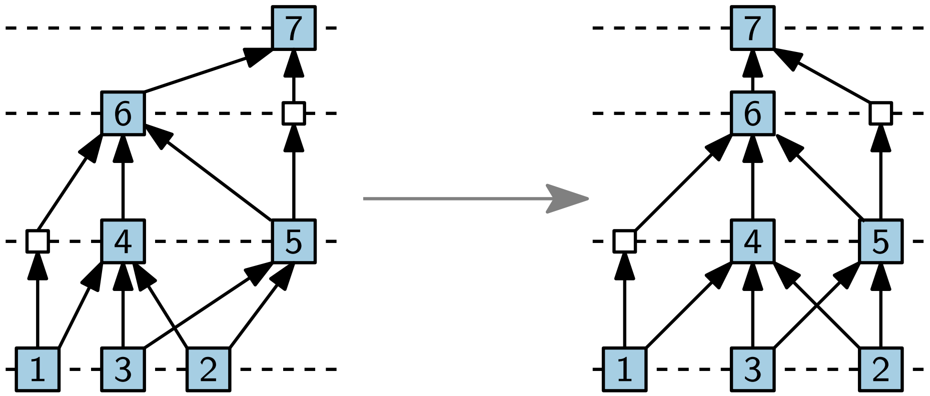
Iterations on Example



Step 4: Vertex Positioning



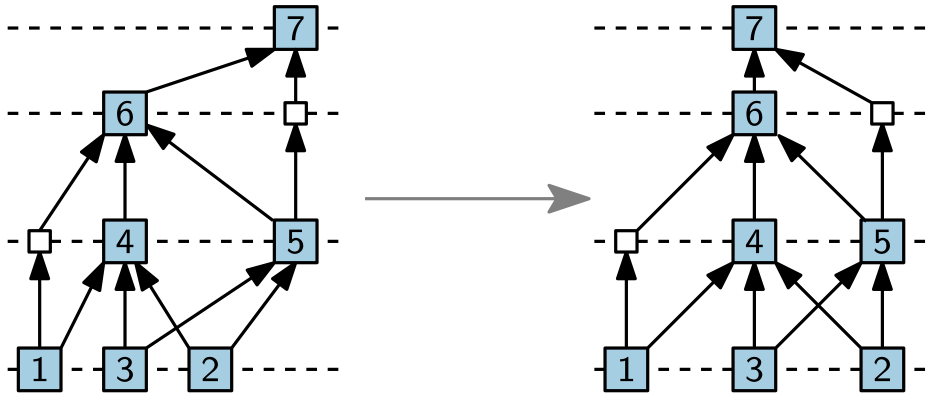
Step 4: Vertex Positioning



Step 4: Vertex Positioning

Goals.

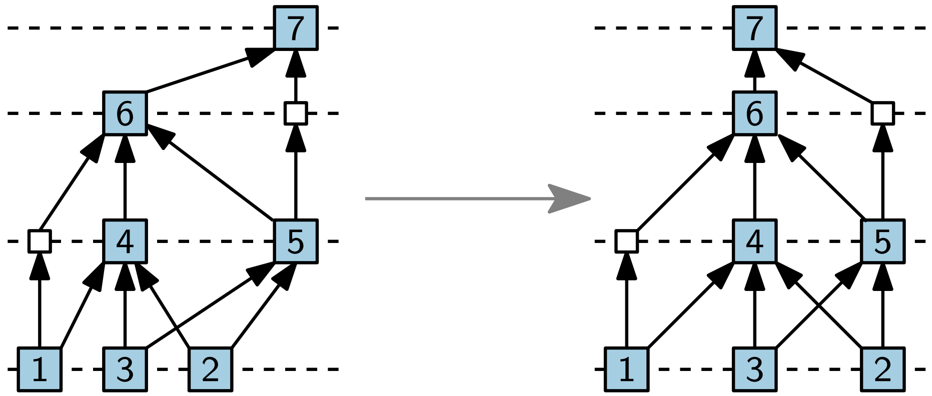
- paths of a single edge should be (close to) straight
- vertices on a layer evenly spaced
- perfer vertical edges



Step 4: Vertex Positioning

Goals.

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

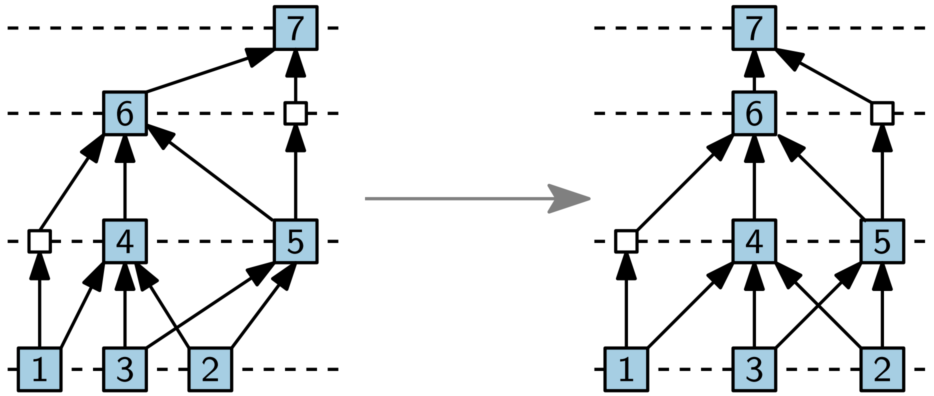


- **Exact:** Quadratic Program (QP)

Step 4: Vertex Positioning

Goals.

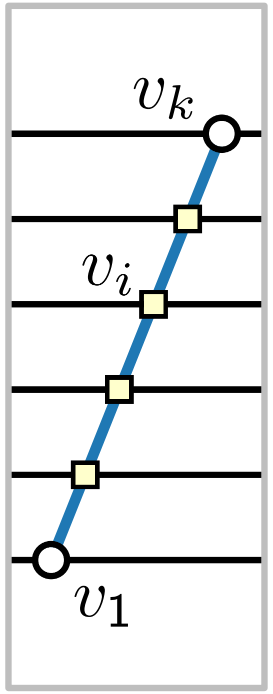
- paths of a single edge should be (close to) straight
- vertices on a layer evenly spaced
- perfer vertical edges



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

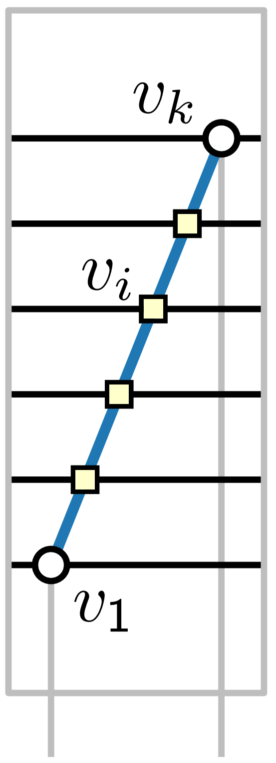
Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.



Quadratic Program

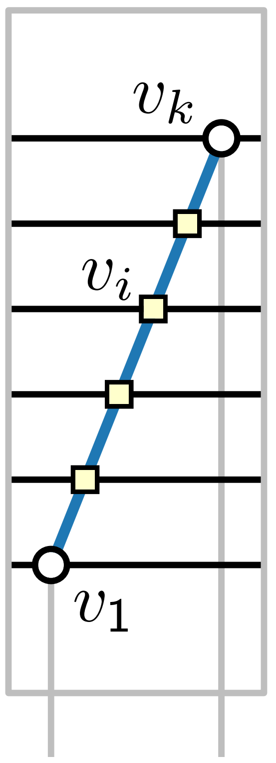
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):



Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

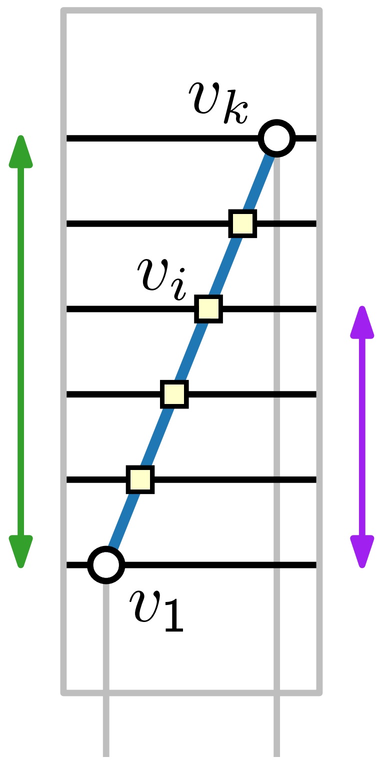
$$\overline{x(v_i)} =$$



Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

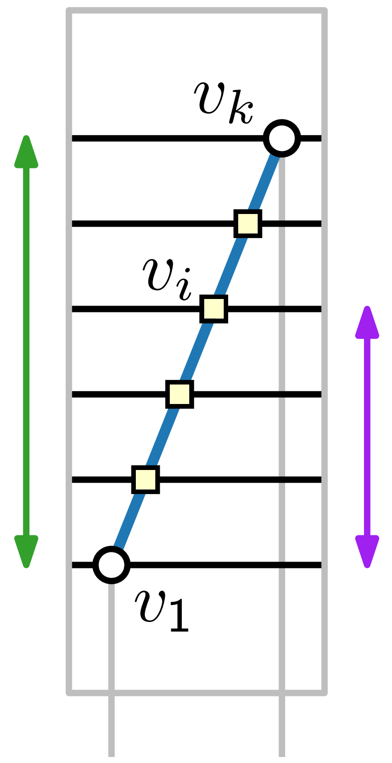
$$\overline{x(v_i)} = x(v_1) +$$



Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

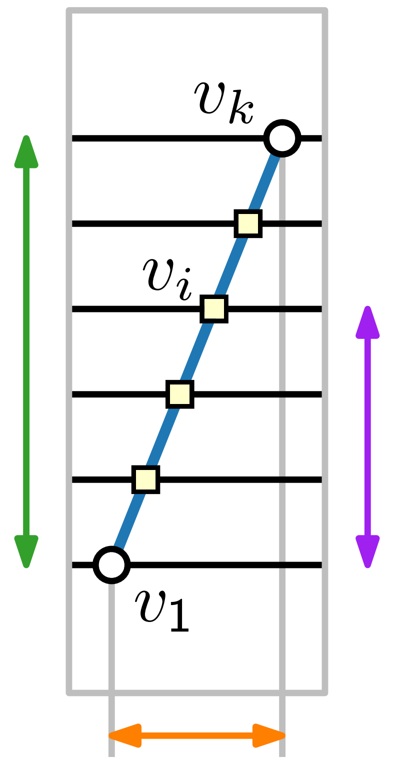
$$\overline{x(v_i)} = x(v_1) + \frac{i-1}{}$$



Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

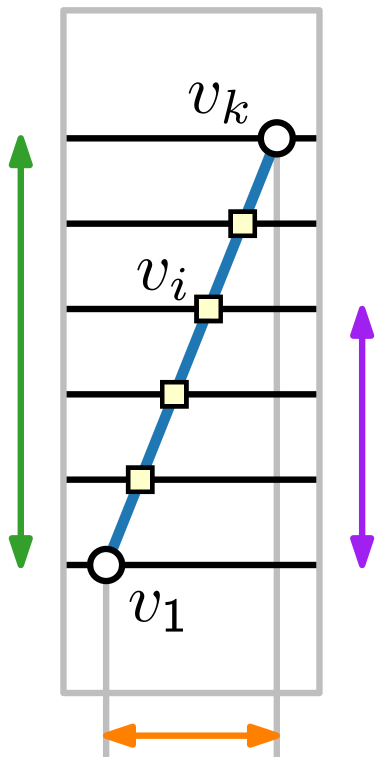
$$\overline{x(v_i)} = x(v_1) + \frac{i-1}{k-1}$$



Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

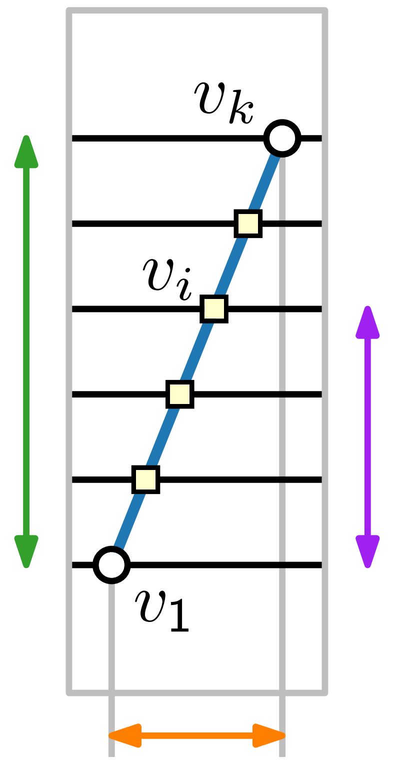


Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- Define the deviation from the line



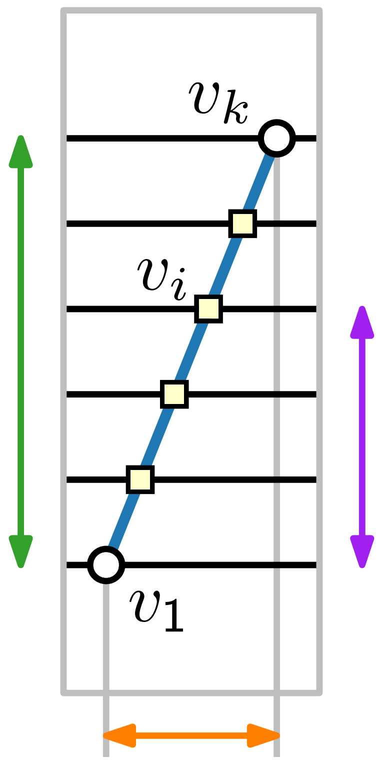
Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- Define the deviation from the line

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



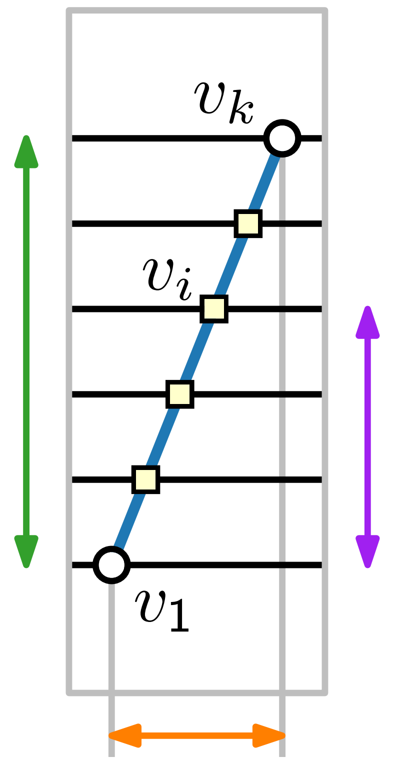
Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- Define the deviation from the line

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



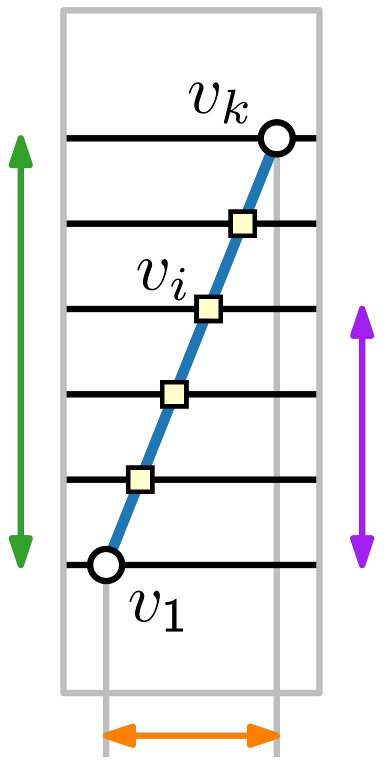
Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- Define the deviation from the line

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



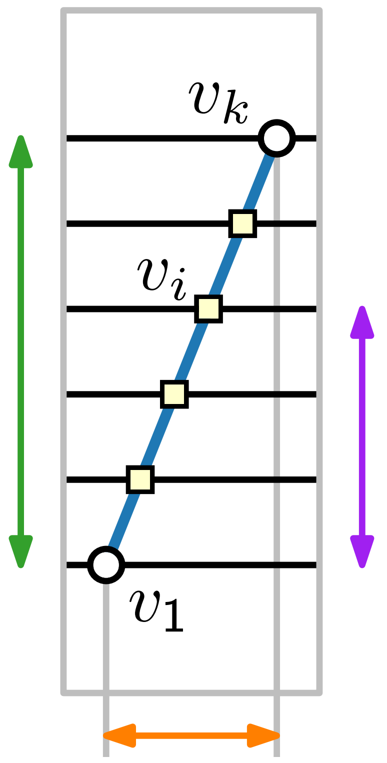
Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- Define the deviation from the line

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



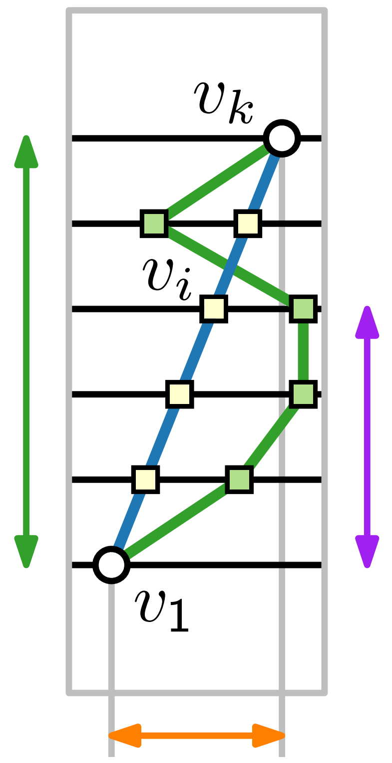
Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- Define the deviation from the line

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



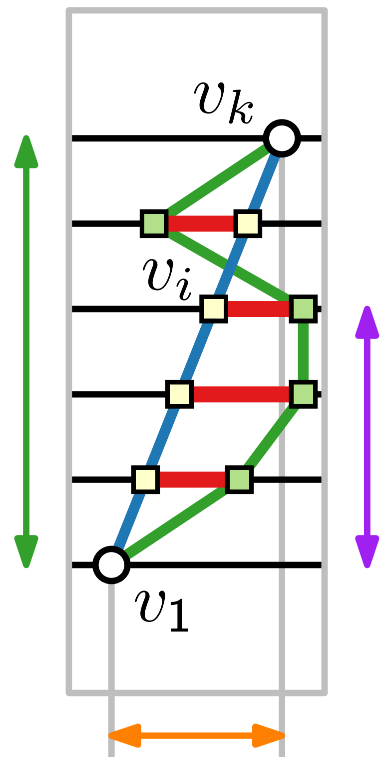
Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- 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

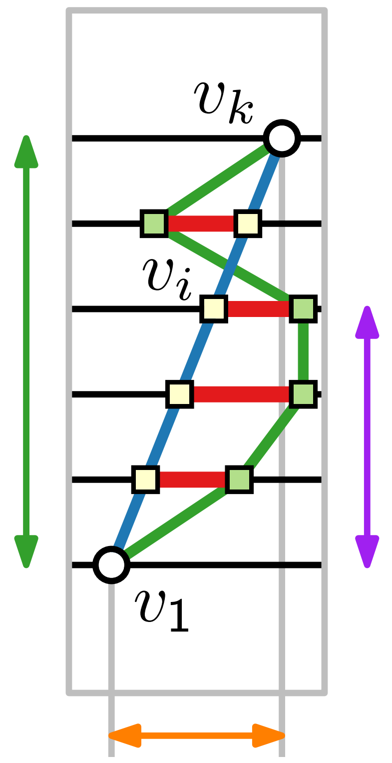
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- 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:



Quadratic Program

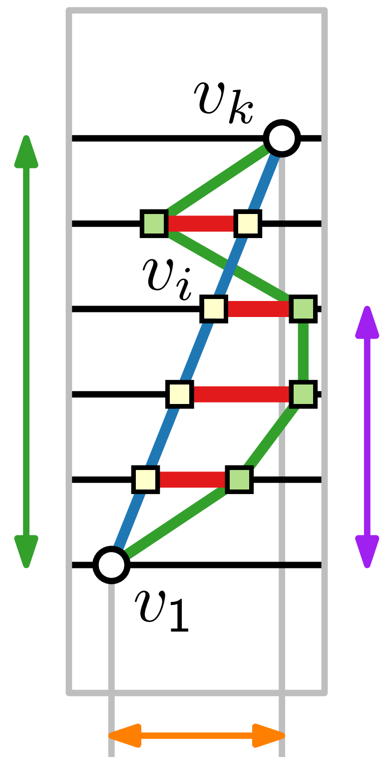
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- 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

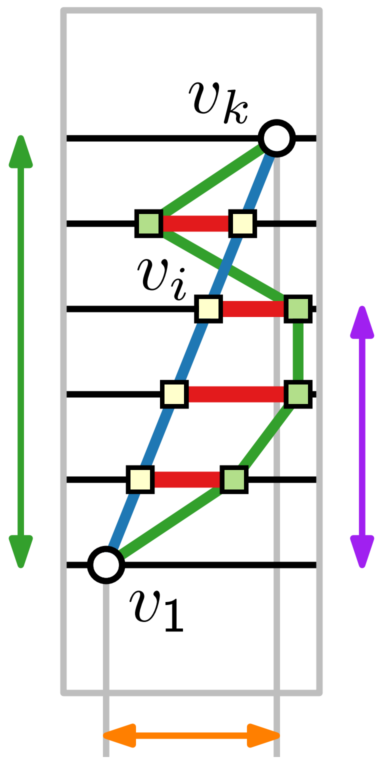
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

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



Quadratic Program

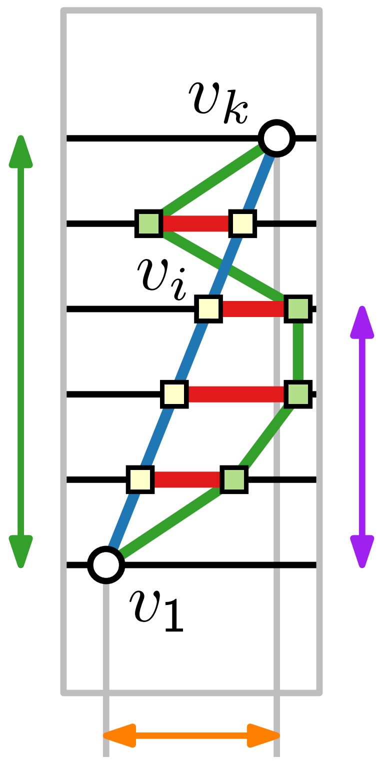
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- Define the deviation from the line

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

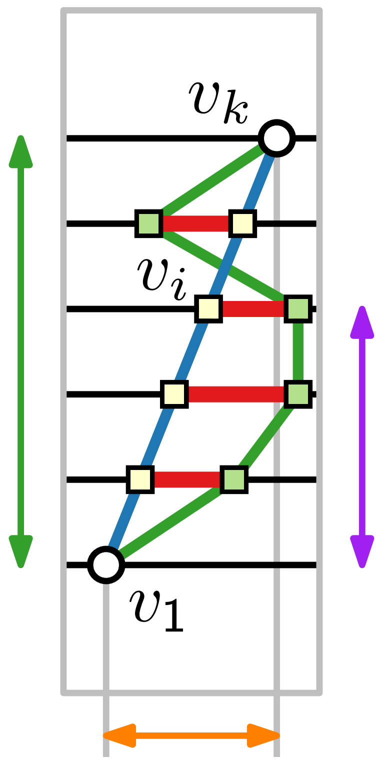
- Objective function: $\min \sum_{e \in E} \mathrm{dev}(p_e)$
- Constraints for all vertices v, w in the same layer with w to the right of v: $x(w) - x(v) \geq \rho$



Quadratic Program

- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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



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

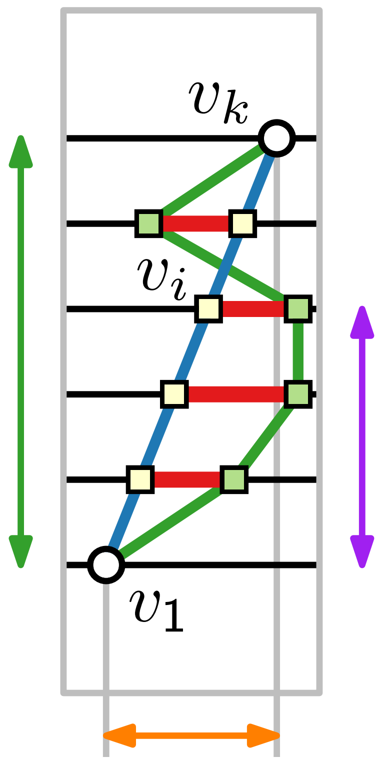
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- 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

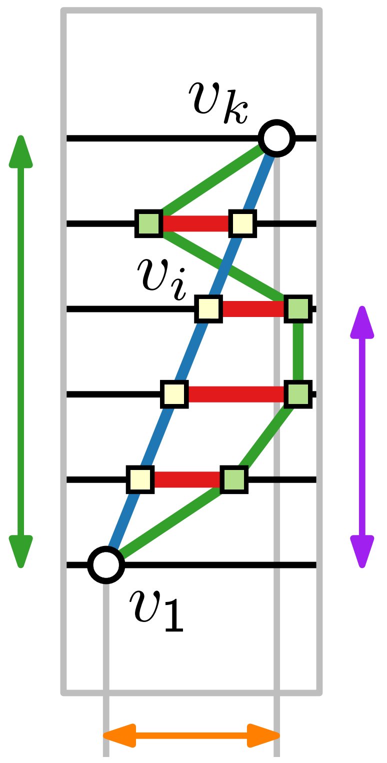
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \ldots, v_k)$ be the corresponding path with dummy vertices $v_2, \ldots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- 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

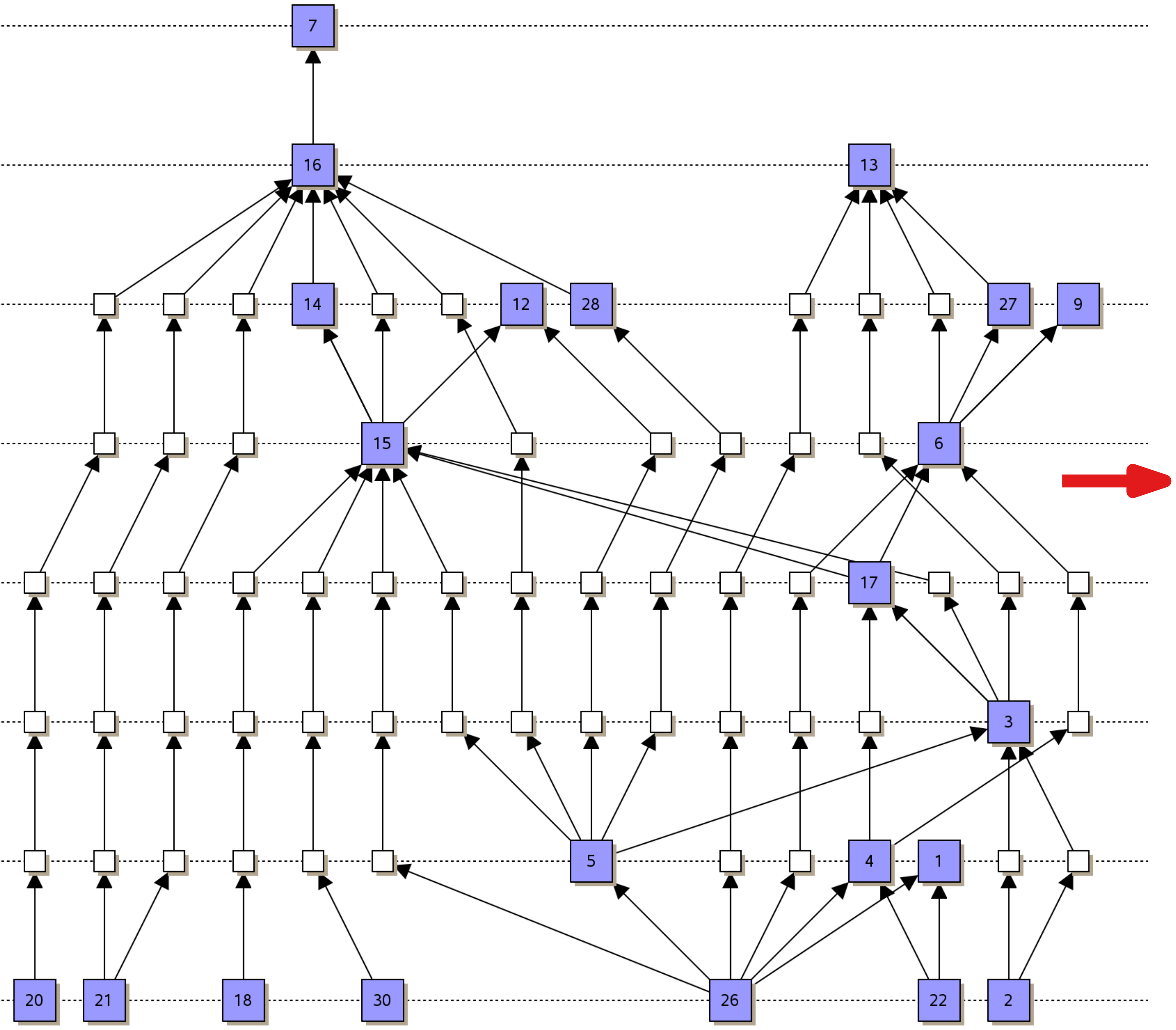
- Apply the following steps as long as improvements can be made:

  1. vertex positioning
  2. edge straightening
  3. compactifying the layout (to reduce the width)

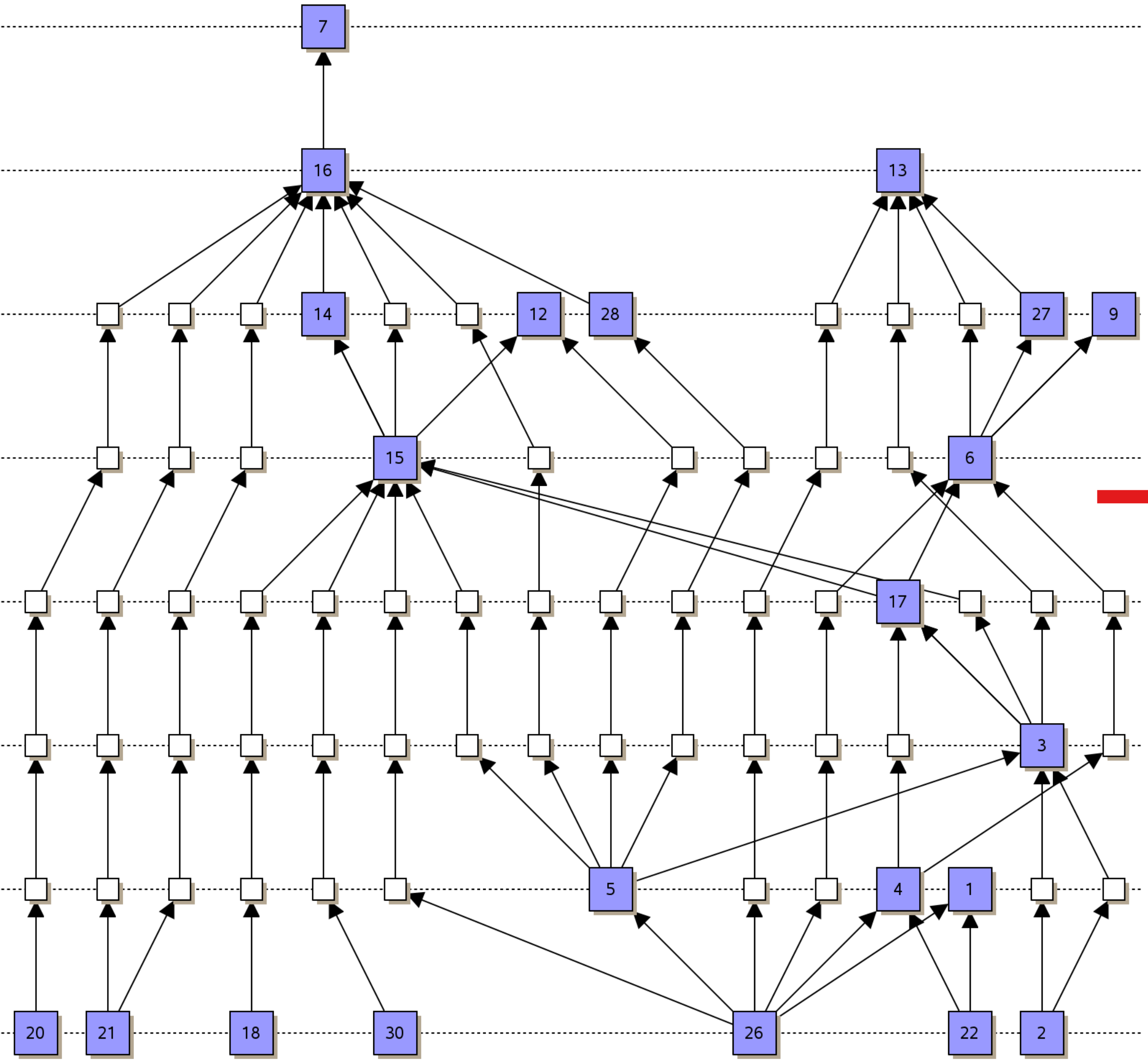
Iterative Heuristic

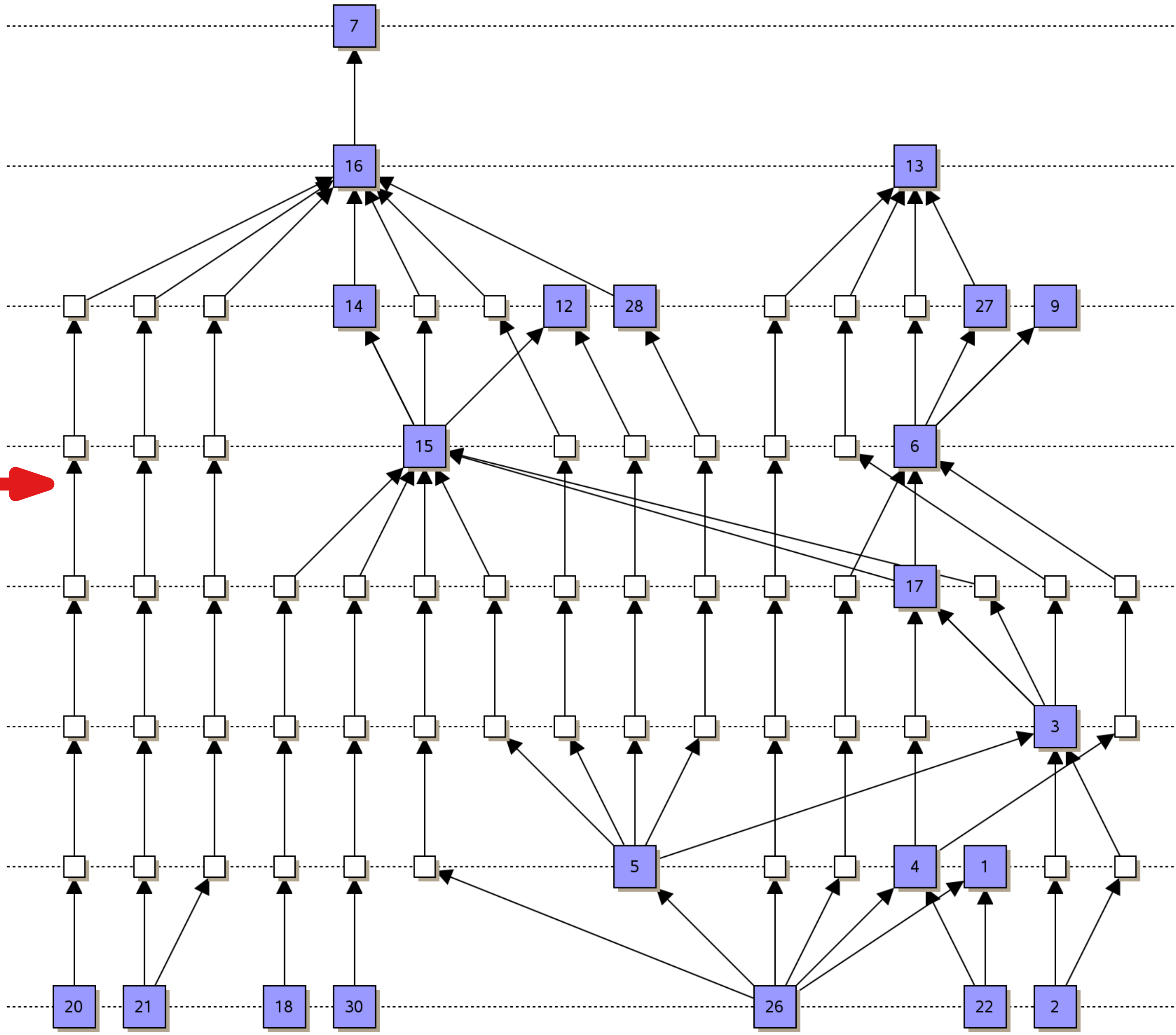
- Compute an initial layout
- Apply the following steps as long as improvements can be made:
  1. vertex positioning
  2. edge straightening
  3. compactifying the layout (to reduce the width)
- Other algorithms, e.g., the one of Brandes and Köpf [GD 2002, see also Brandes, Walter, Zink: arXiv 2020]:
  – tries to align vertices vertically
  – does horizontal compaction afterwards
  – linear running time

Example

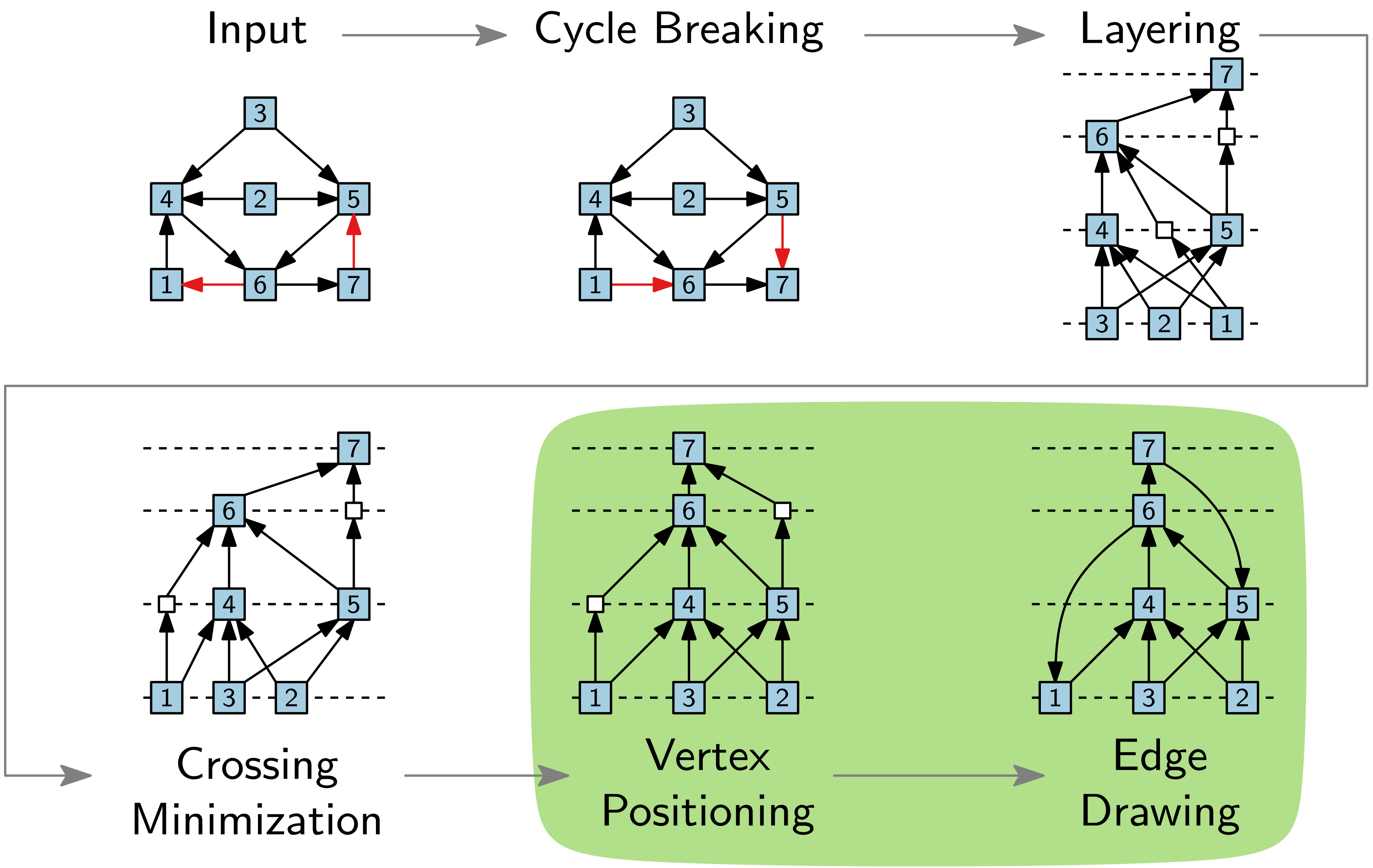


Example


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

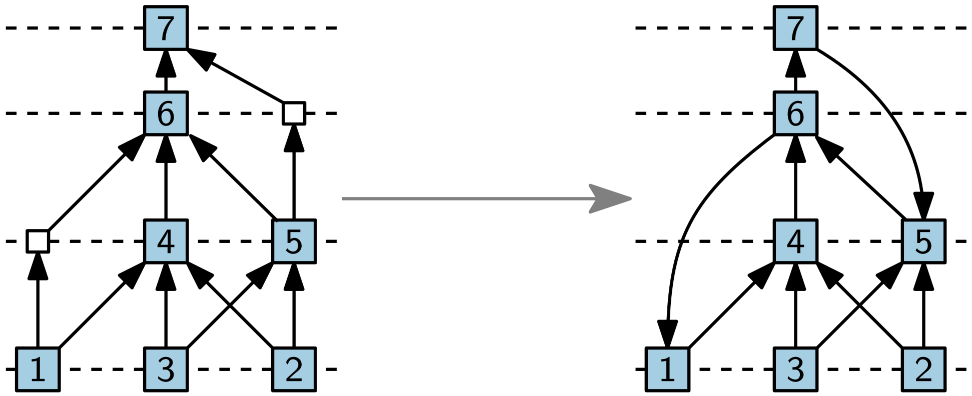

7
16
13
14
12
28
27
9
15
6
17
3
5
4
1
20
21
18
30
26
22
2

Step 5: Drawing Edges



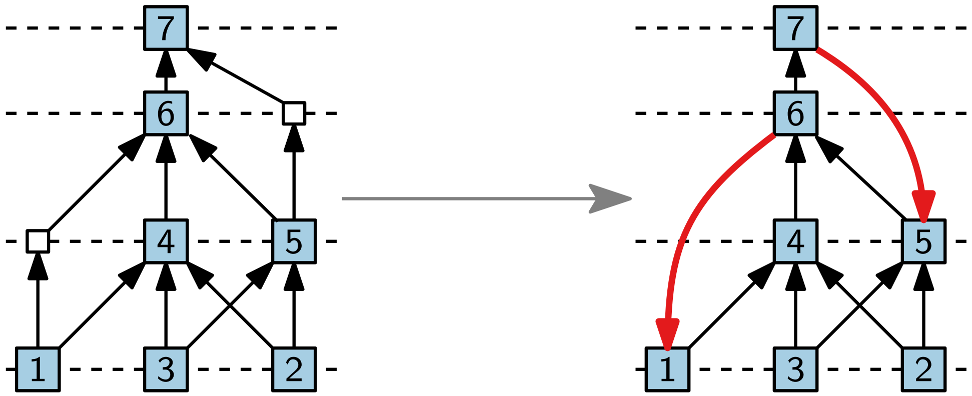
Step 5: Drawing Edges

Possibility.
Substitute polylines by Bézier curves.



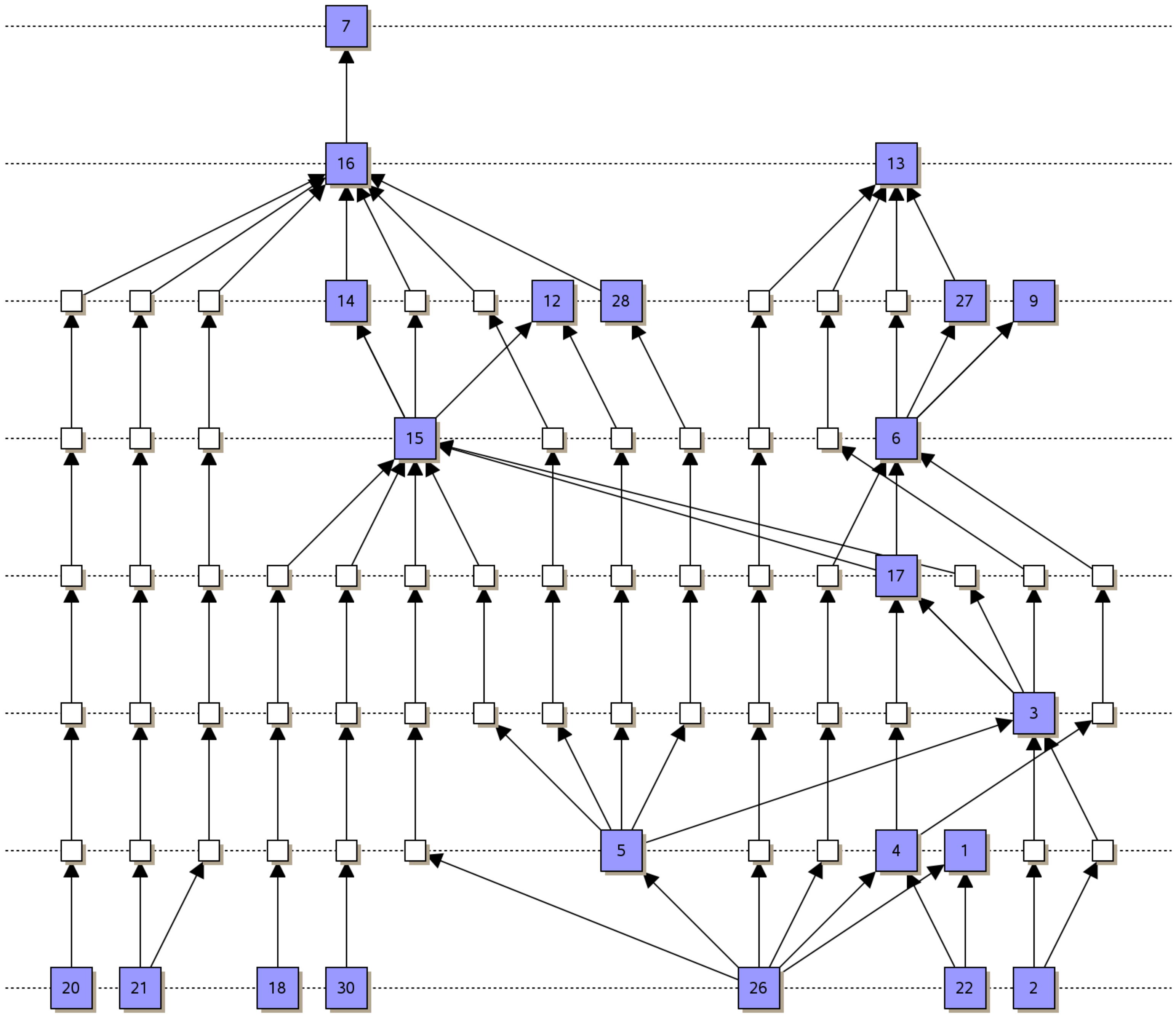
Step 5: Drawing Edges

Possibility.
Substitute polylines by Bézier curves.

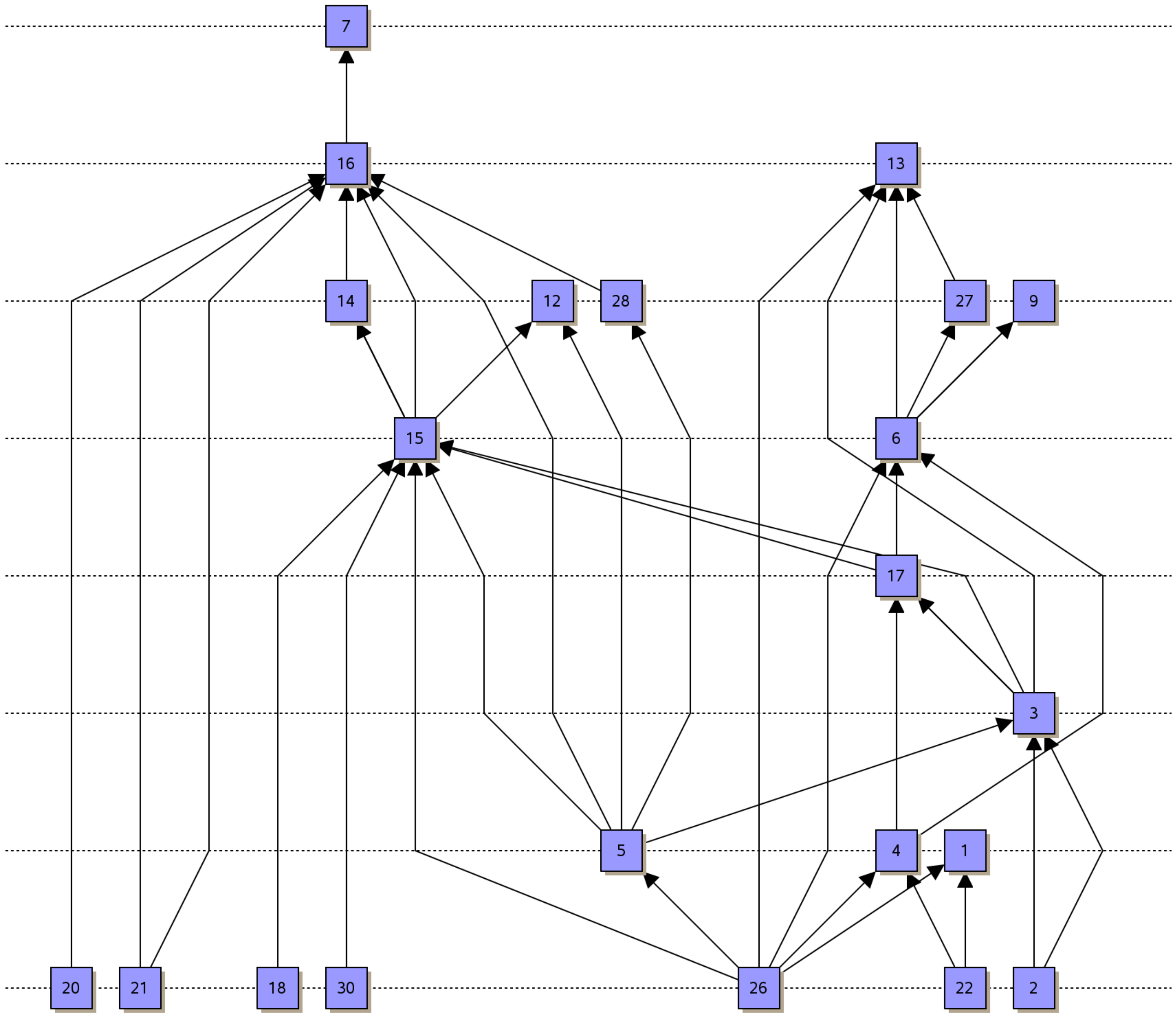


Remark.
Draw reversed edges downwards.

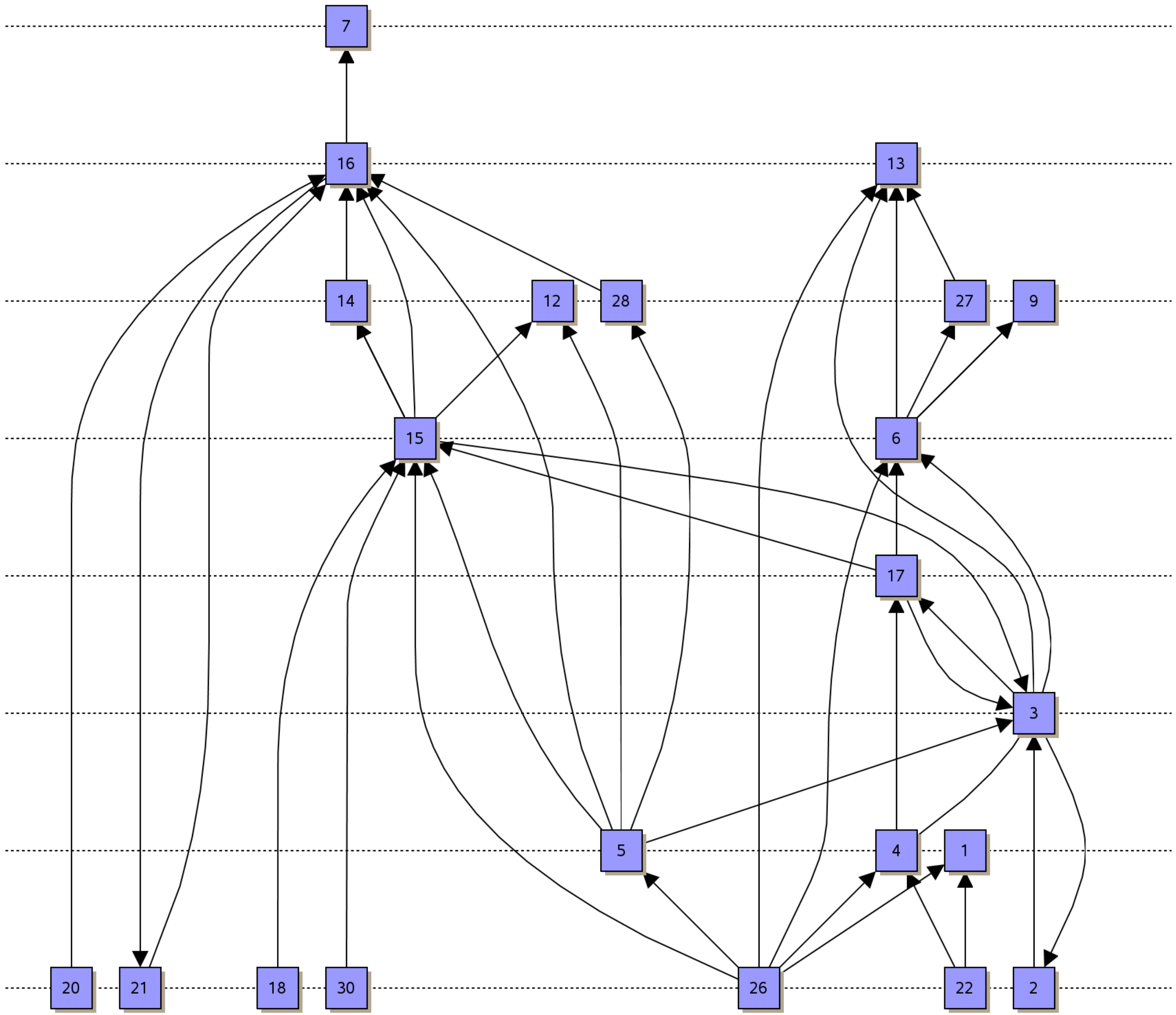
Example



Example

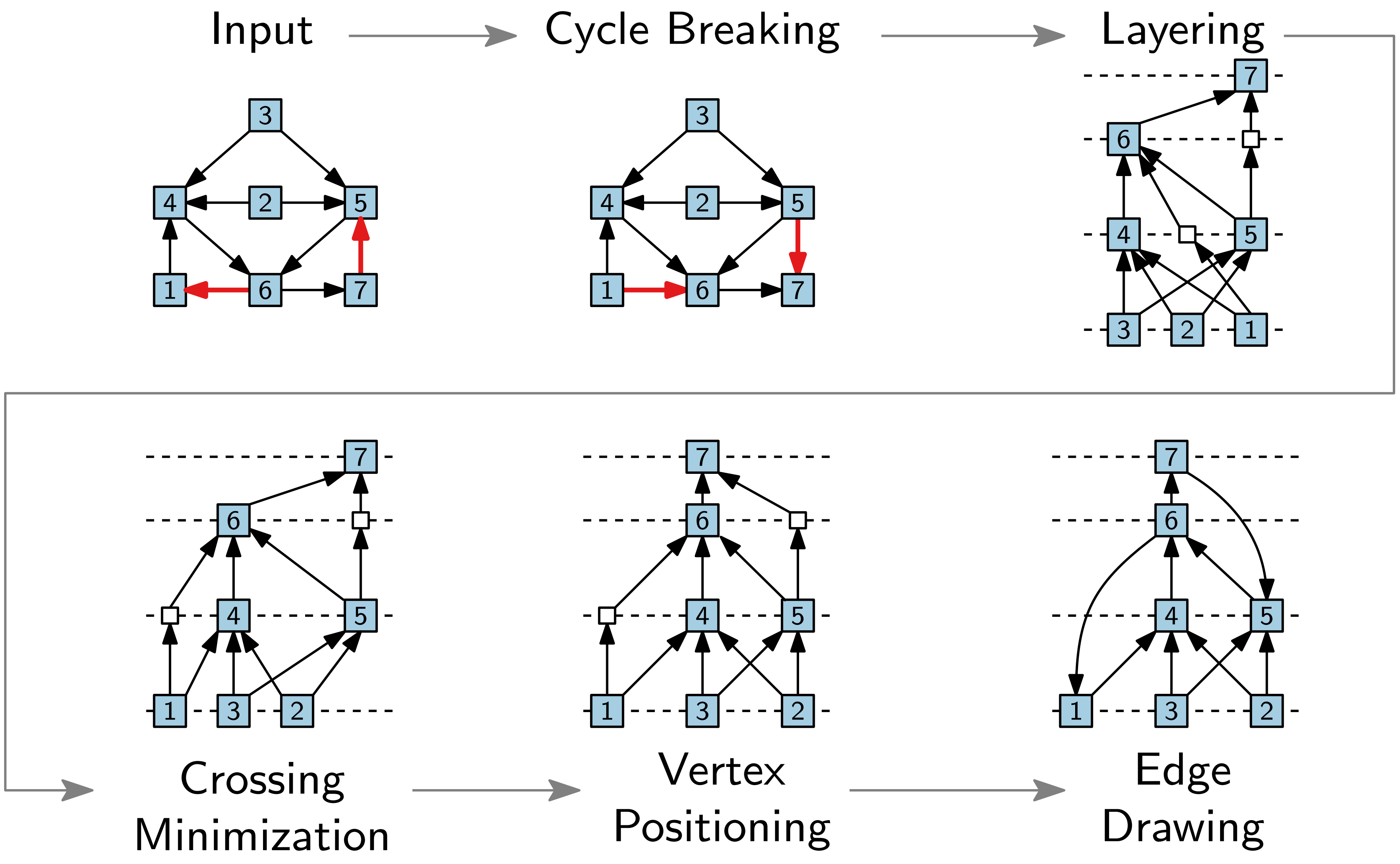


Example



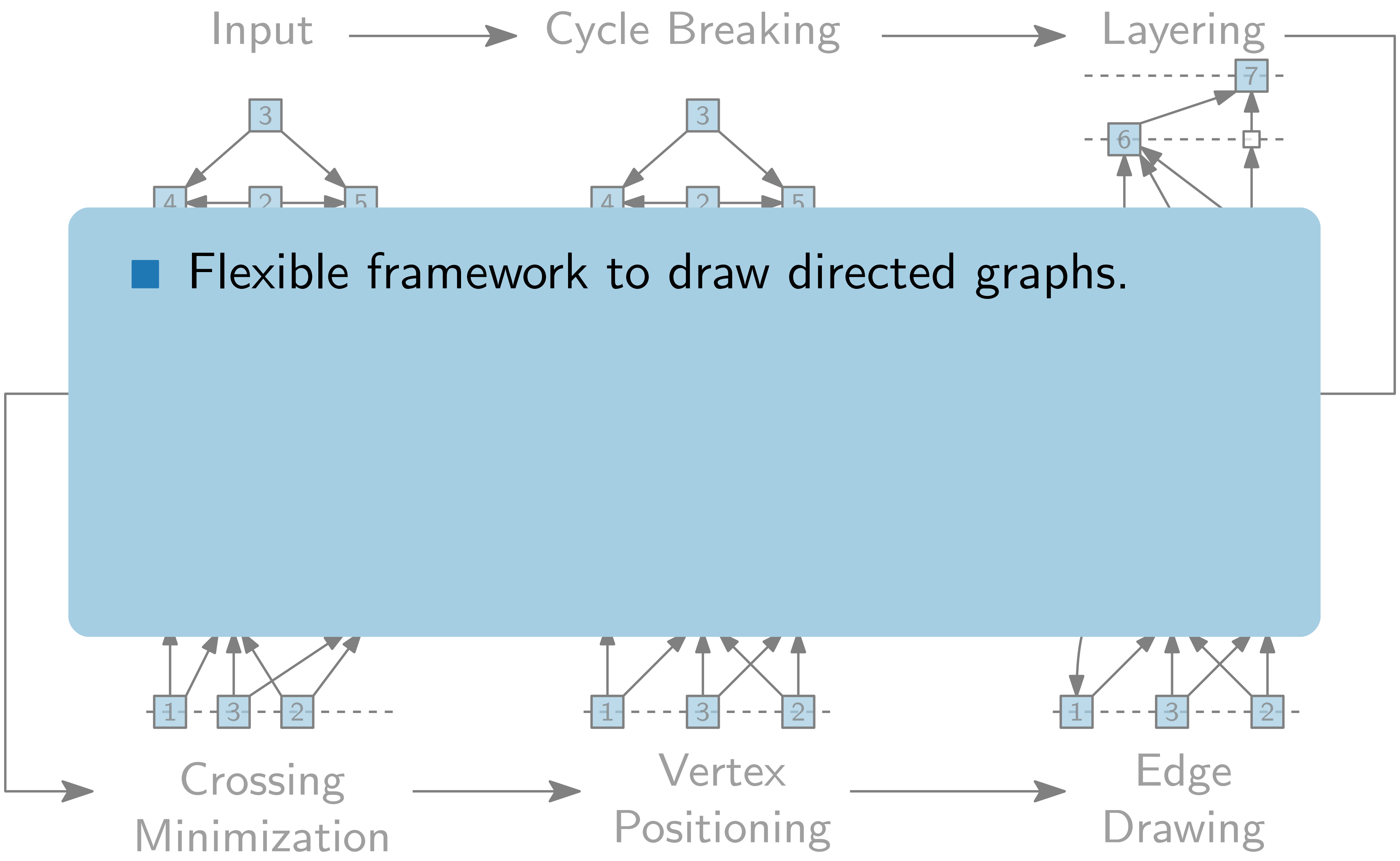
Classical Approach – Sugiyama Framework

[Sugiyama, Tagawa, Toda '81]



Classical Approach – Sugiyama Framework

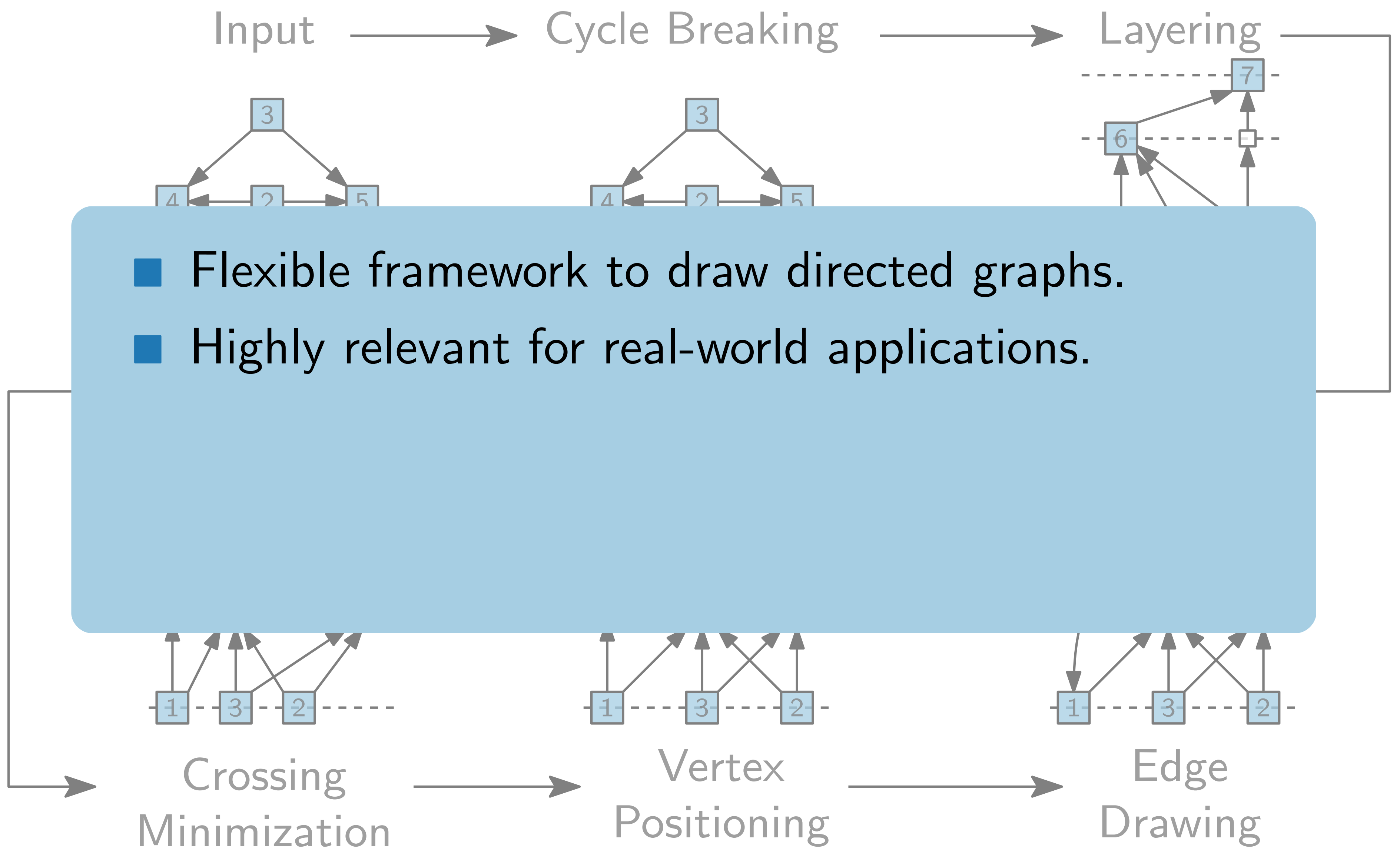
[Sugiyama, Tagawa, Toda '81]



- Flexible framework to draw directed graphs.

Classical Approach – Sugiyama Framework

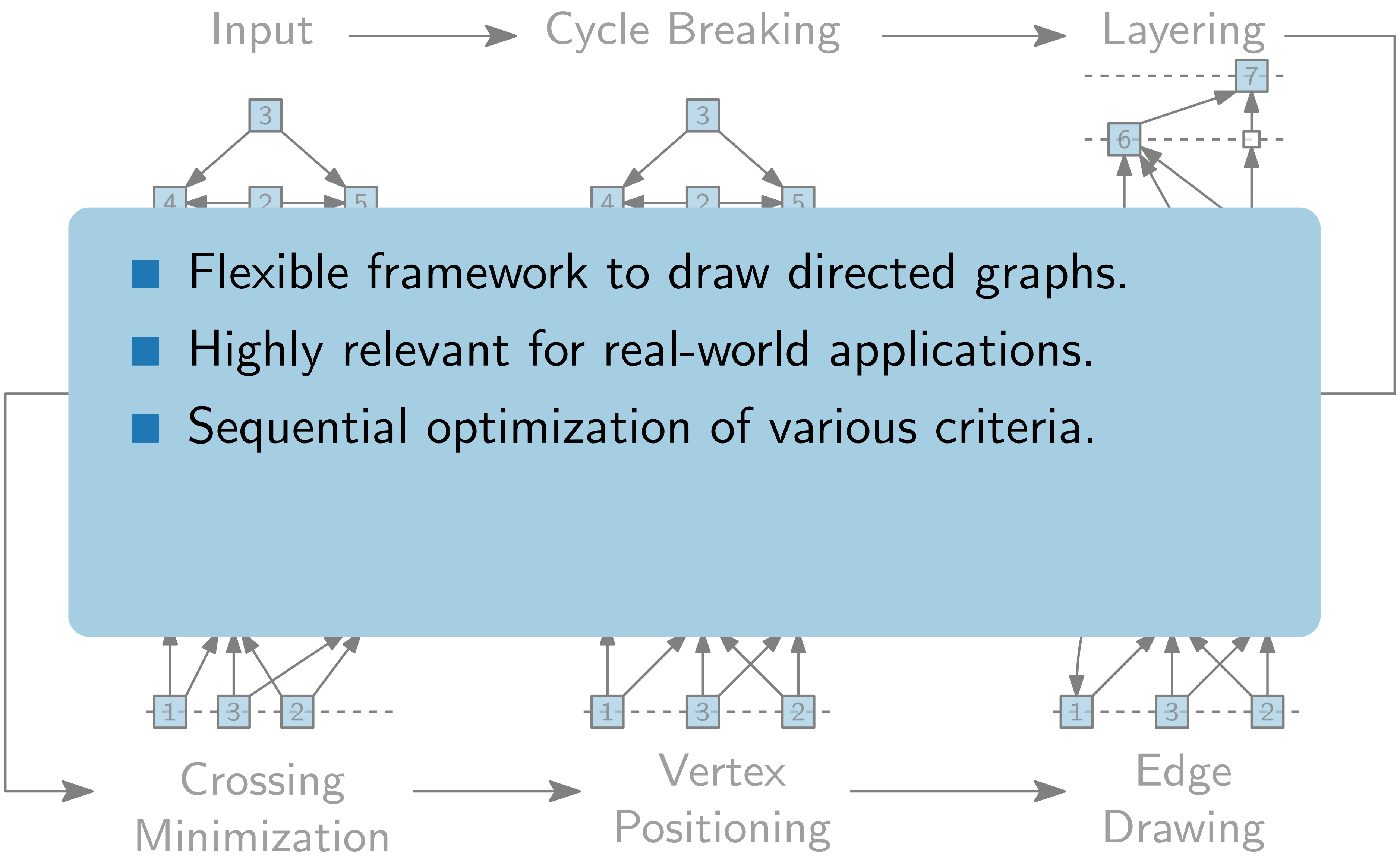
[Sugiyama, Tagawa, Toda '81]



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

Classical Approach – Sugiyama Framework

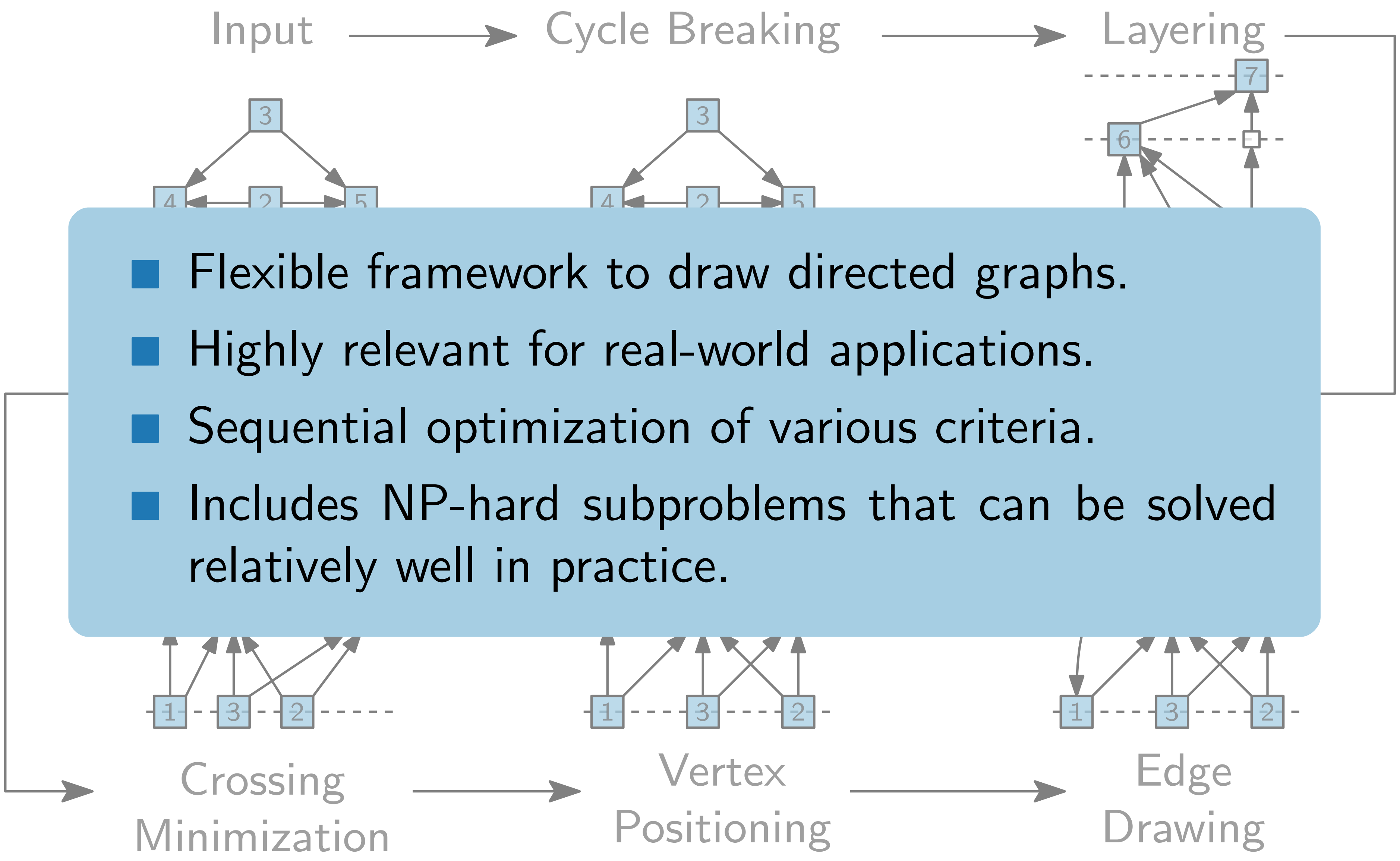
[Sugiyama, Tagawa, Toda '81]



- 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 subproblems that can be solved relatively 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