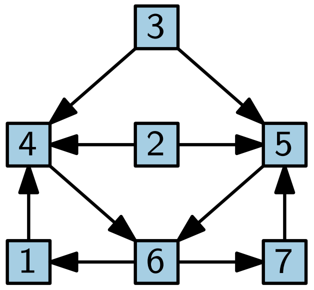
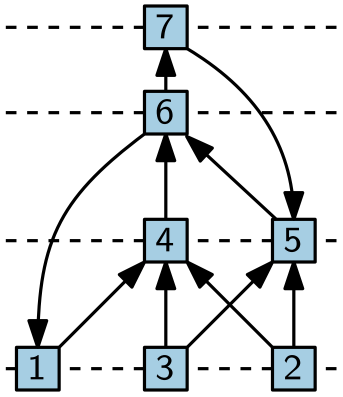
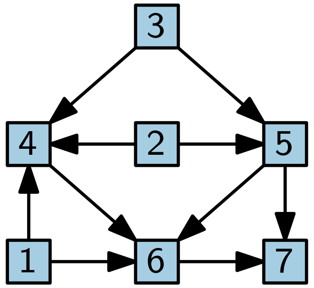
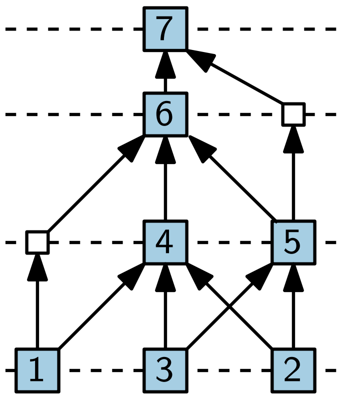
Visualization of Graphs

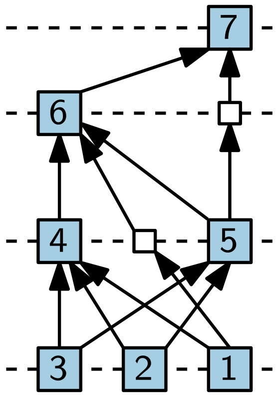




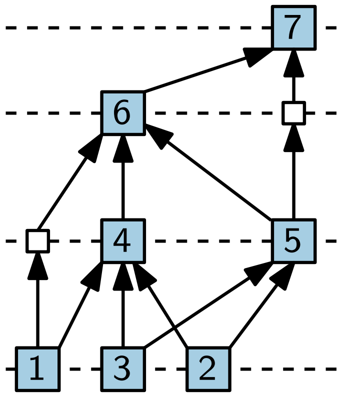
Lecture 8:
Hierarchical Layouts:
Sugiyama Framework





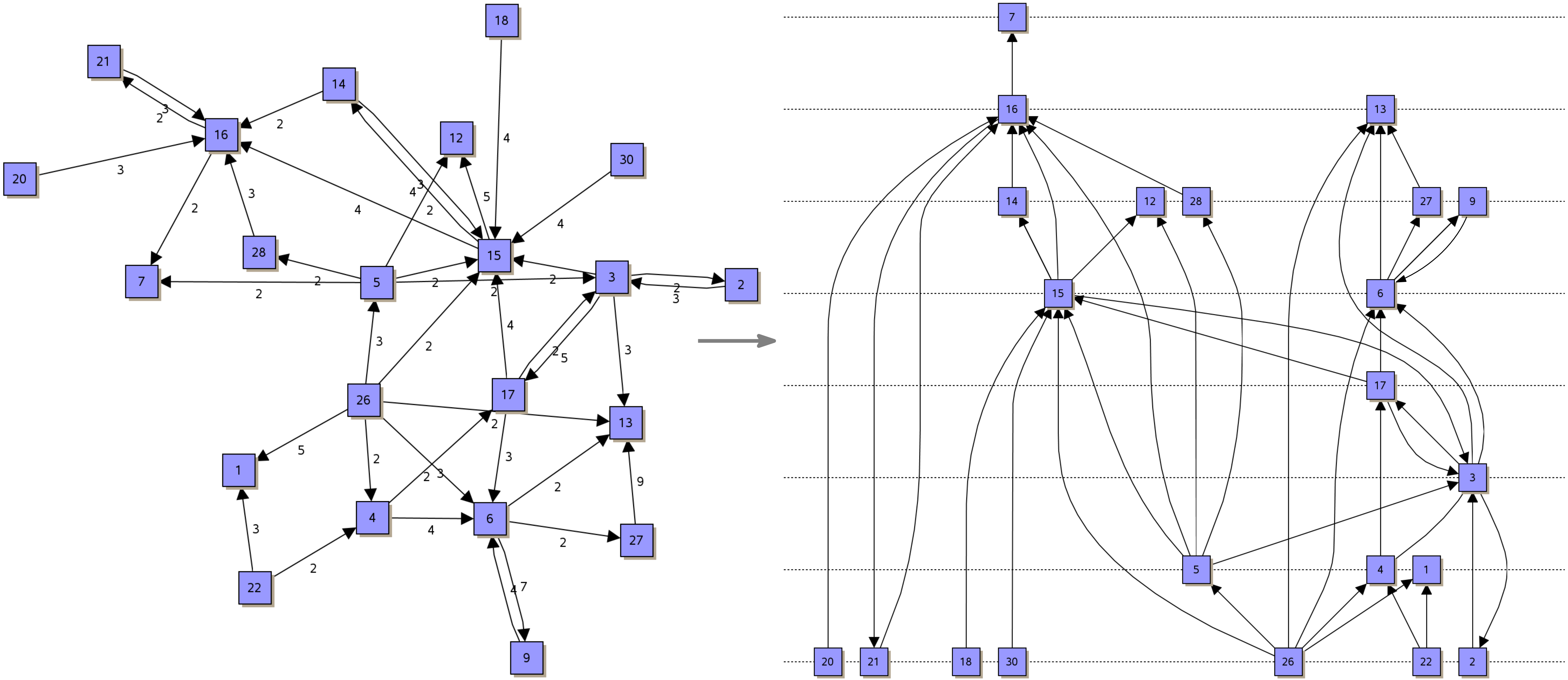


Alexander Wolff



Summer term 2025

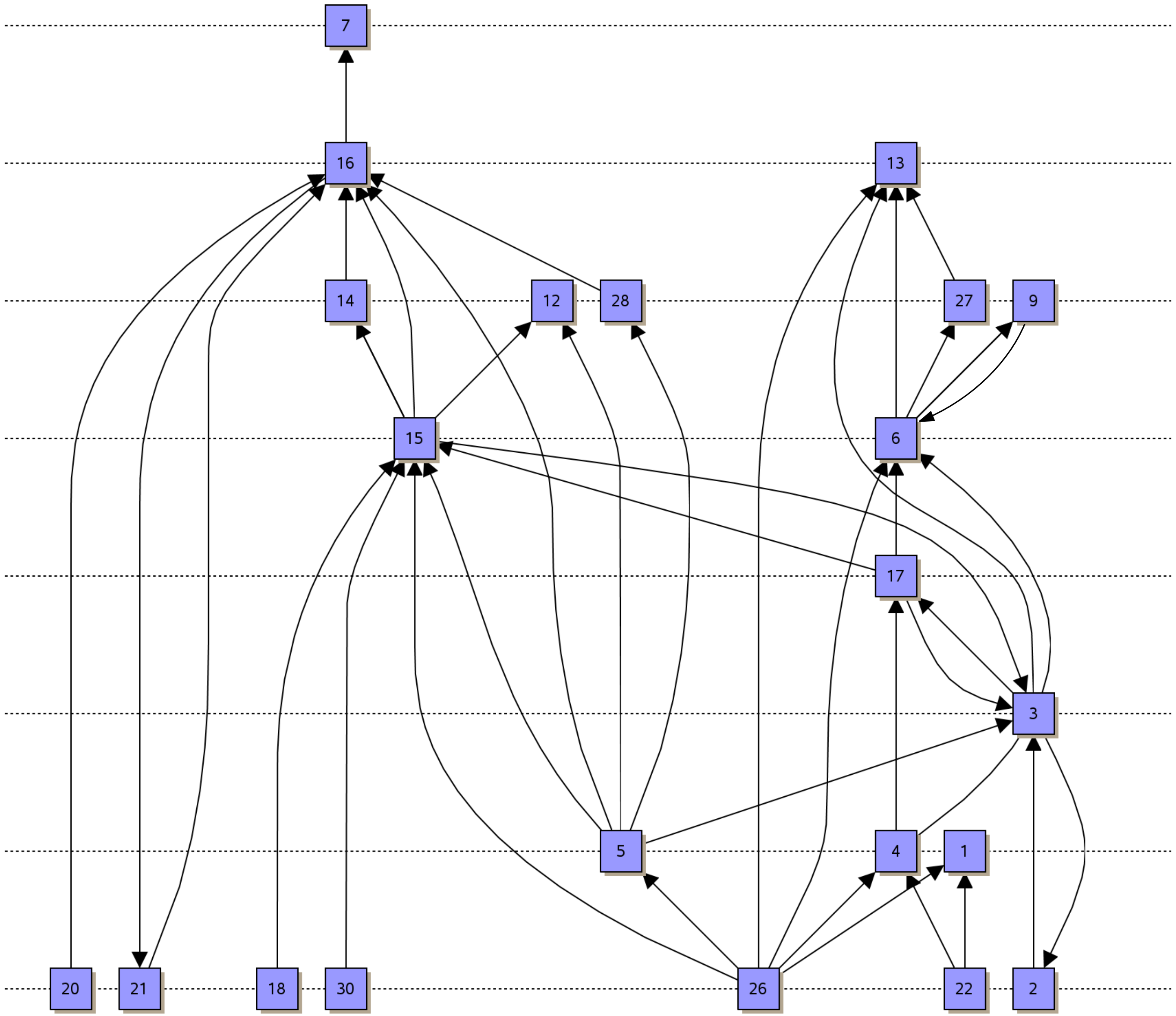
Hierarchical Drawings – Motivation



Hierarchical Drawing

Problem Statement:

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



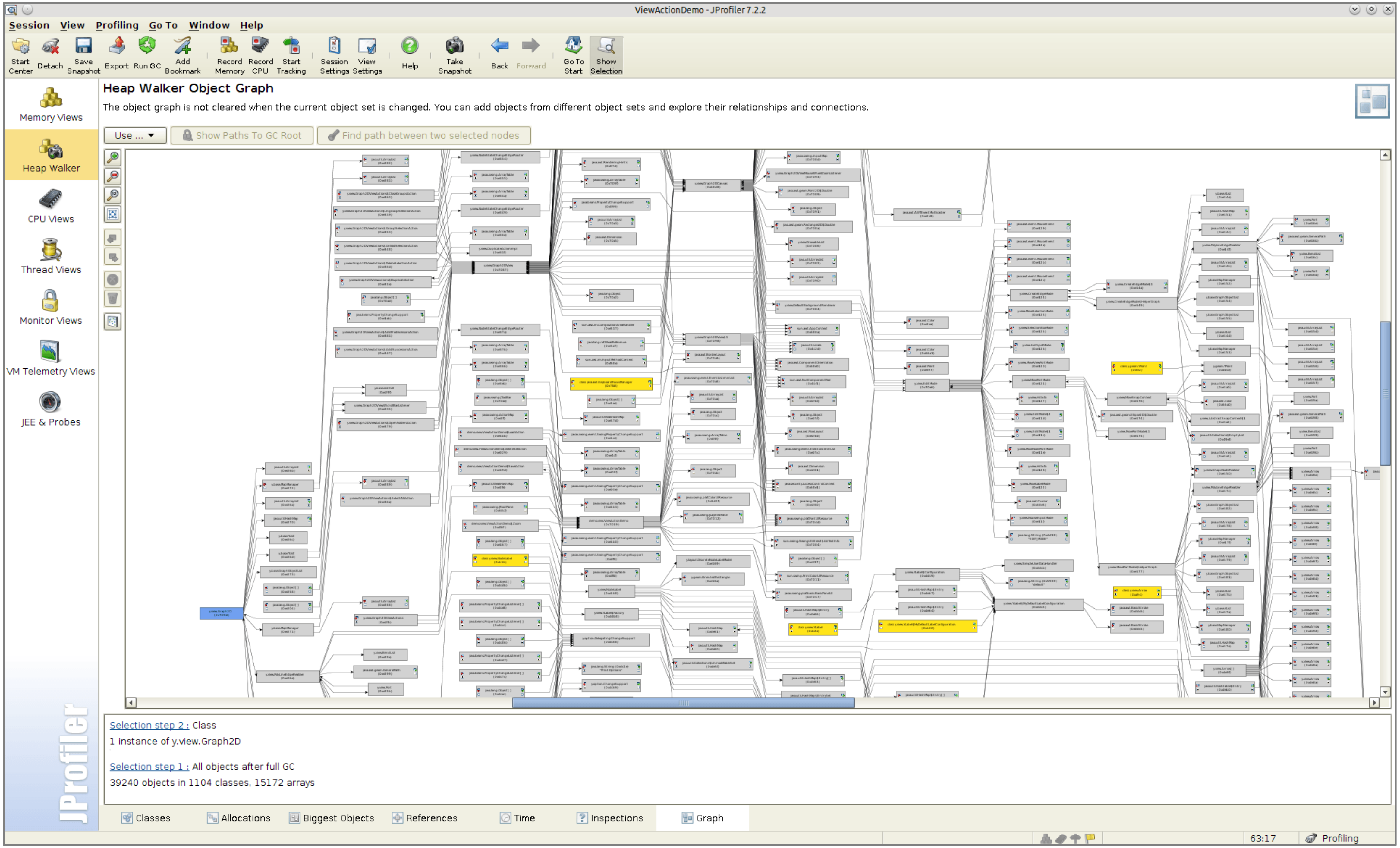
Desirable Properties:

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

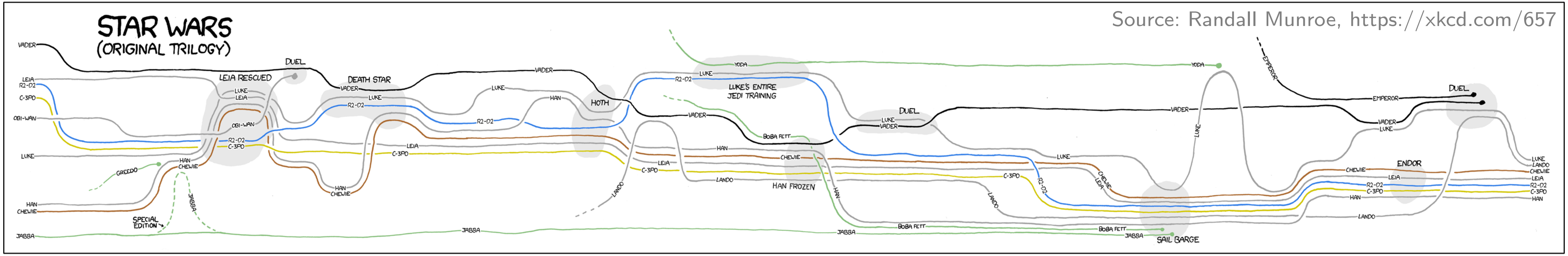
Criteria can be contradictory!

Hierarchical Drawing – Applications

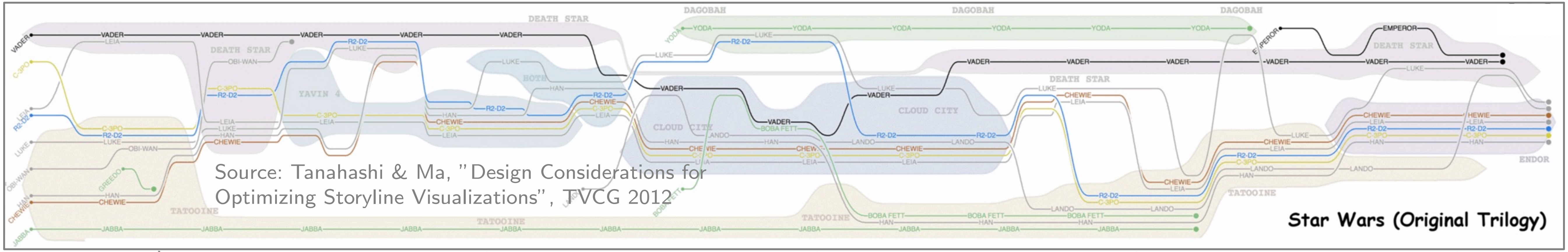
yEd Gallery: Java profiler JProfiler using yFiles



Hierarchical Drawing – Applications

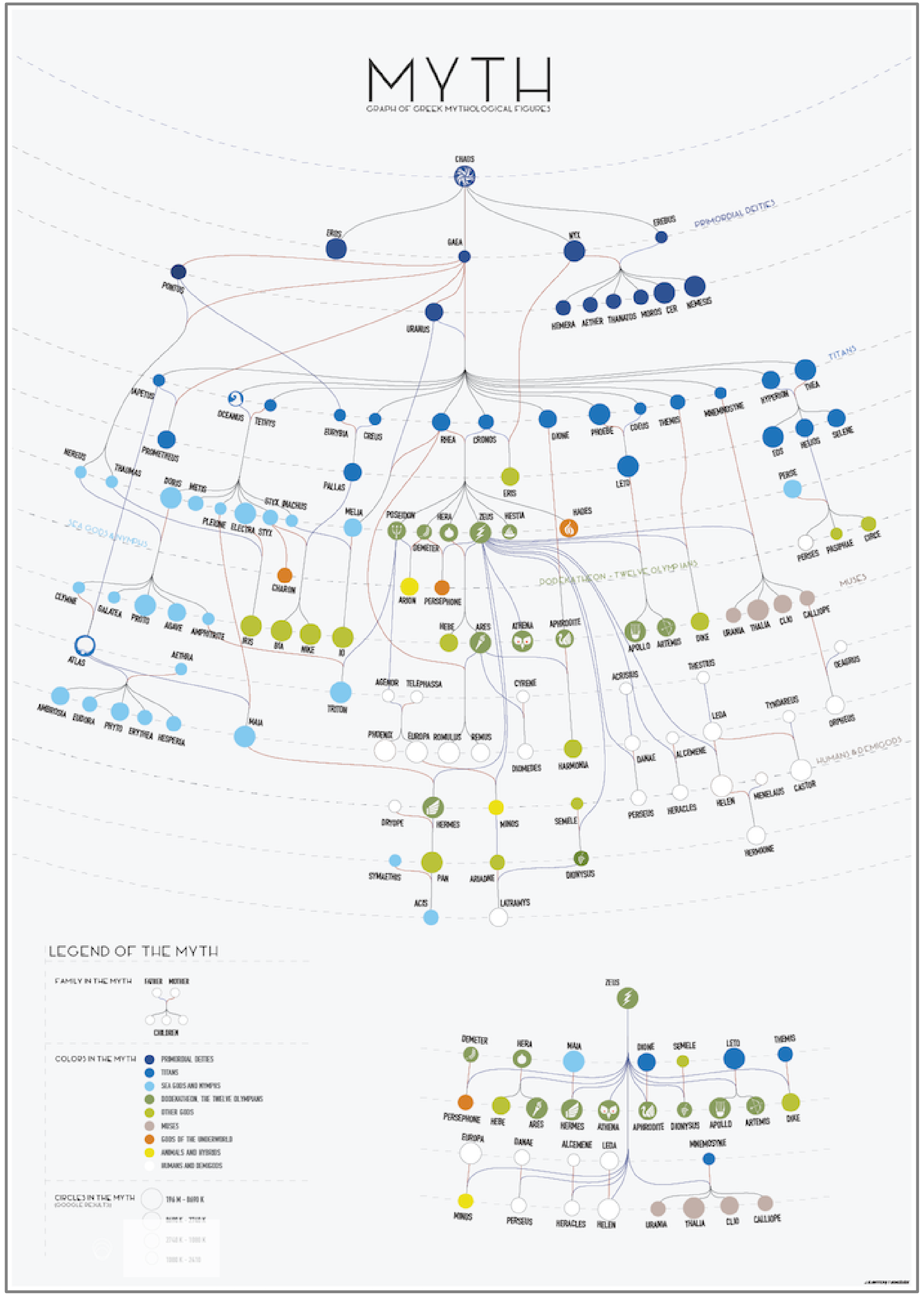


Source: Randall Munroe, https://xkcd.com/657



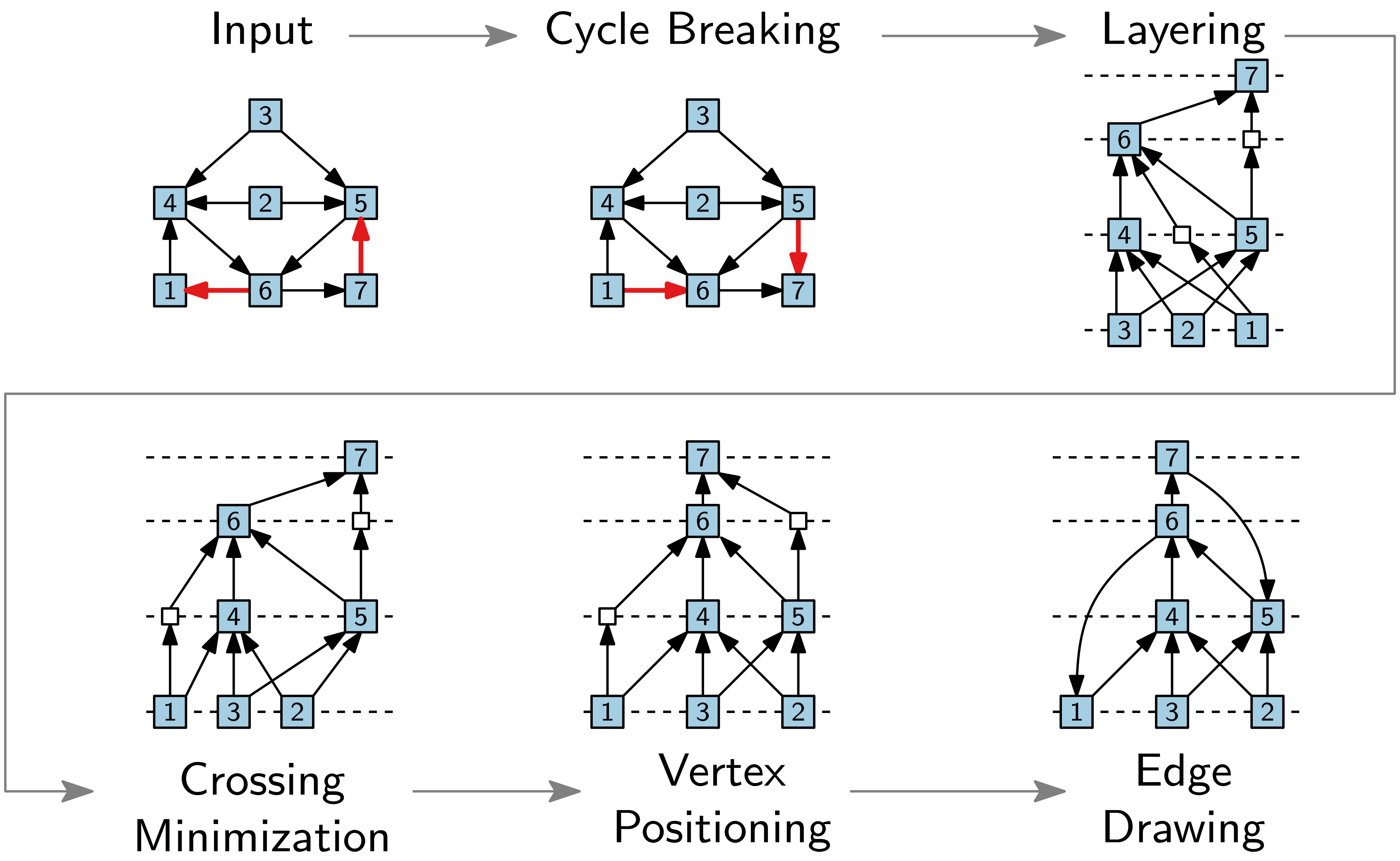
Source: Tanahashi & Ma, "Design Considerations for Optimizing Storyline Visualizations", TVCG 2012

Hierarchical Drawing – Applications

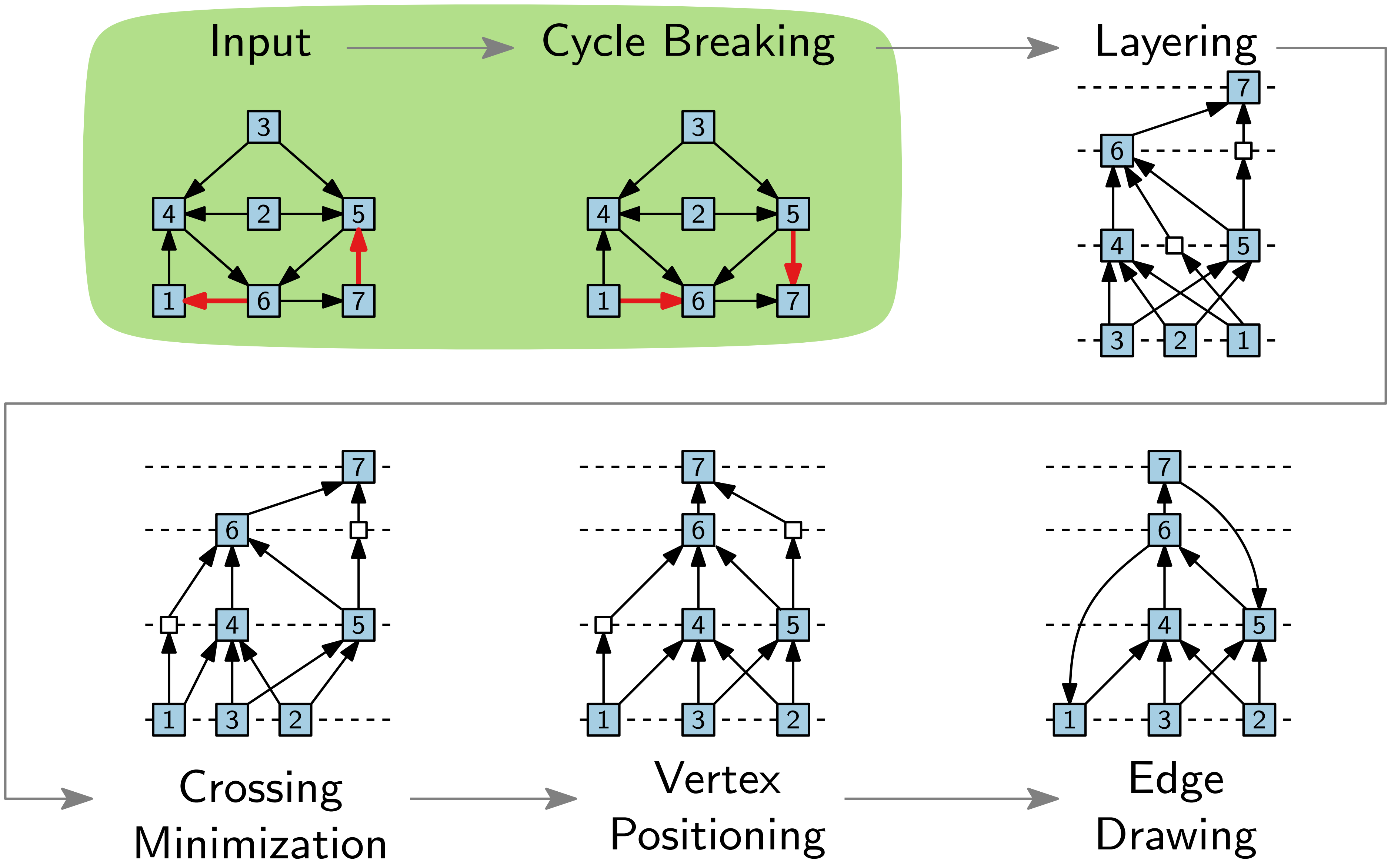


Source: Visualization that won the Graph Drawing Contest, Creative Track, 2016. Klawitter & Mchedlidze

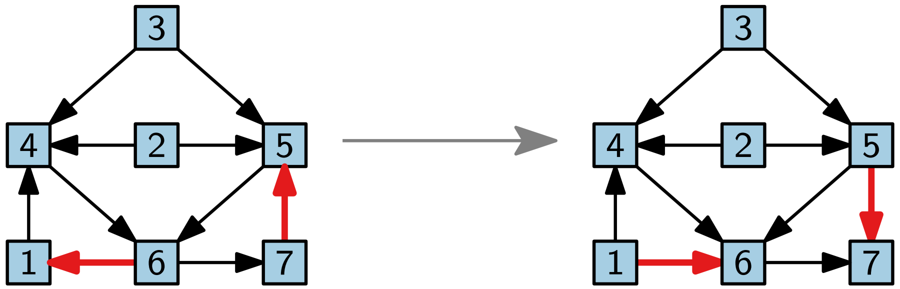
Classical Approach: Sugiyama Framework [Sugiyama, Tagawa, Toda ’81]



Step 1: Cycle Breaking



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

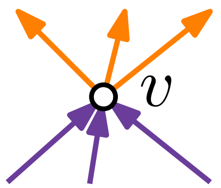
$(E(G) \setminus E^\star) \cup E^\star_{\text{rev}}$

. . . NP-hard ☹

Heuristic 1

[Berger, Shor '90]

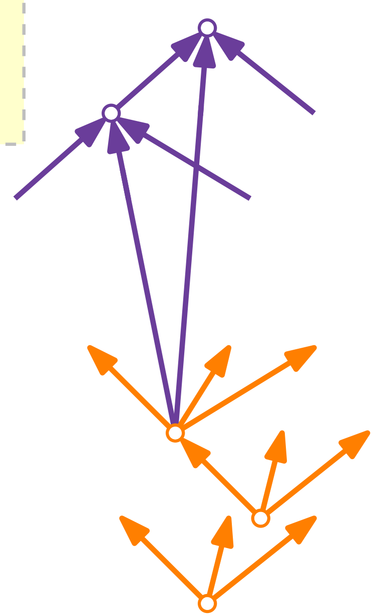
GreedyMakeAcyclic(Digraph G):
- $E' \leftarrow \emptyset$
- **foreach** $v \in V(G)$ **do**
  - **if** $|E^{\rightarrow}(v)| \geq |E^{\leftarrow}(v)|$ **then**
    - $E' \leftarrow E' \cup E^{\rightarrow}(v)$
  - **else**
    - $E' \leftarrow E' \cup E^{\leftarrow}(v)$
  - remove v and $E(v)$ from G.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

Proof Idea.



Place the vertices on distinct y-coordinates.

y-coordinates increase/decrease towards the middle.

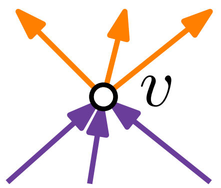
All edges point upwards.

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.

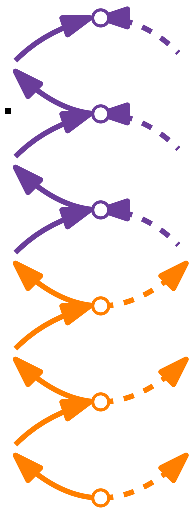
Proof Idea.

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**

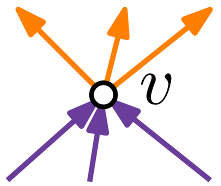
    $E' \leftarrow E' \cup E^{\rightarrow}(v)$

  else

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

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

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



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

- 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]

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

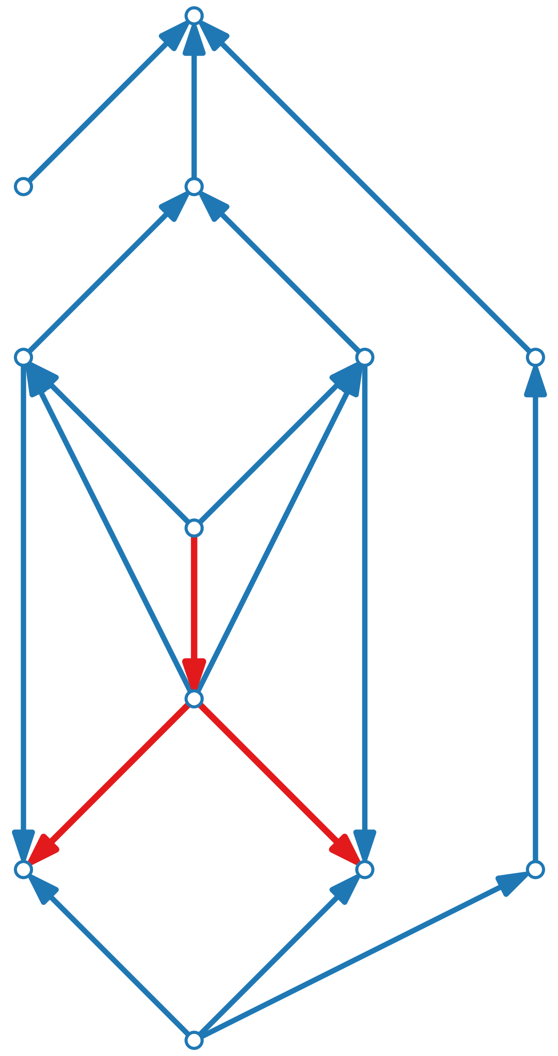
  if $V(G) \neq \emptyset$ **then**

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

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

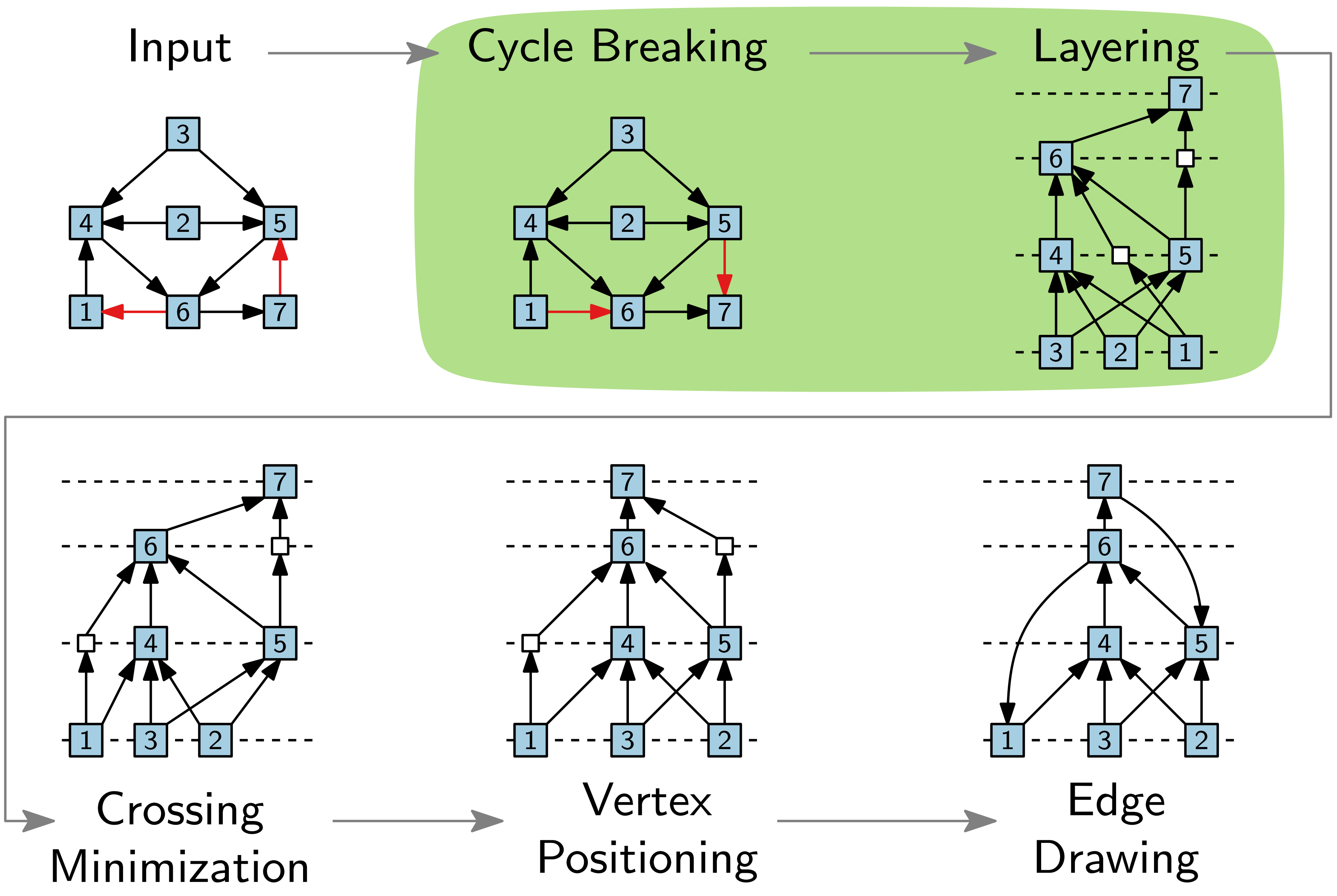
    Remove v and $E(v)$ from G.

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

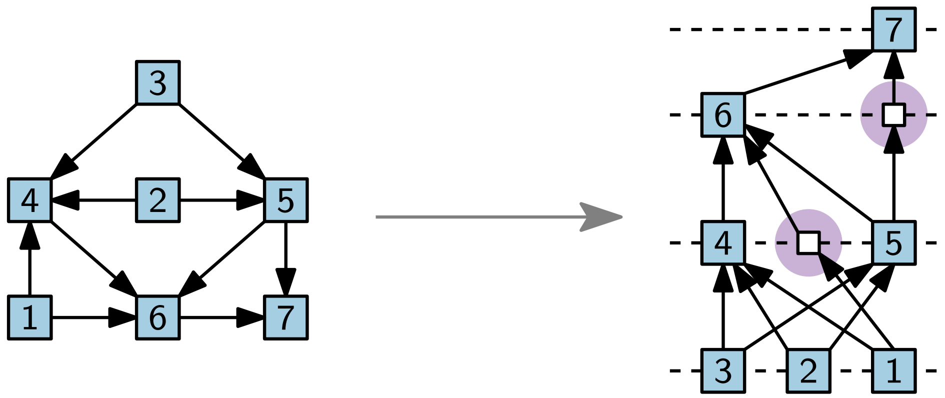


- Time: $\mathcal{O}(|V(G)| + |E(G)|)$ [Main idea: Use bins for sinks and sources, and a bin for each $|N^{\rightarrow}(v)| - |N^{\leftarrow}(v)|$]
- Quality guarantee: $|E'| \geq |E(G)|/2 + |V(G)|/6$

Step 2: Layering



Step 2: Layering



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 $(u, v) \in E(G)$, $y(u) < y(v)$.

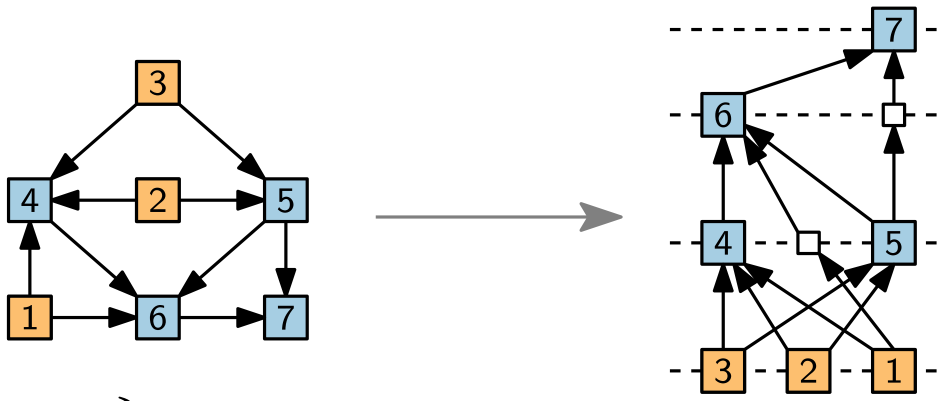
Objective is to *minimize* . . .

- 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.

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

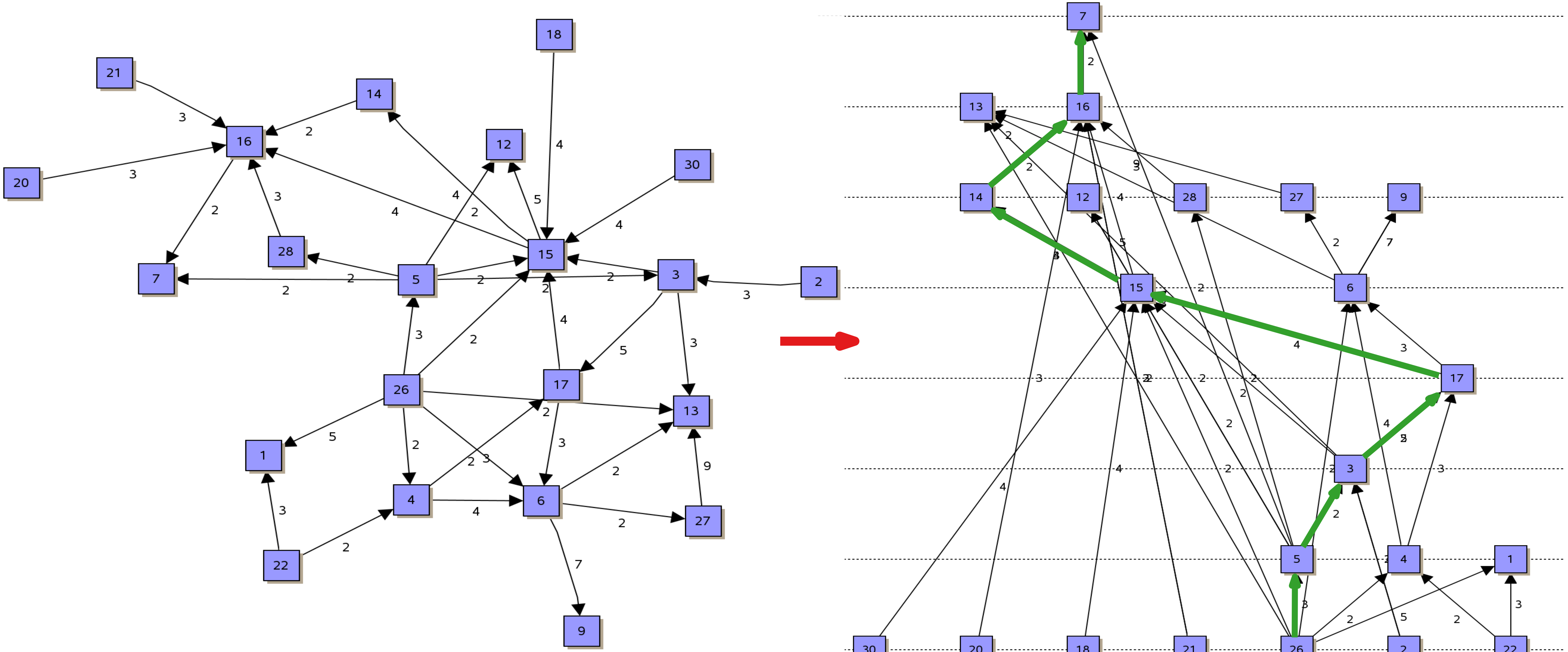


Observation.

- $y(v)$ 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



Minimize Total Edge Length – ILP

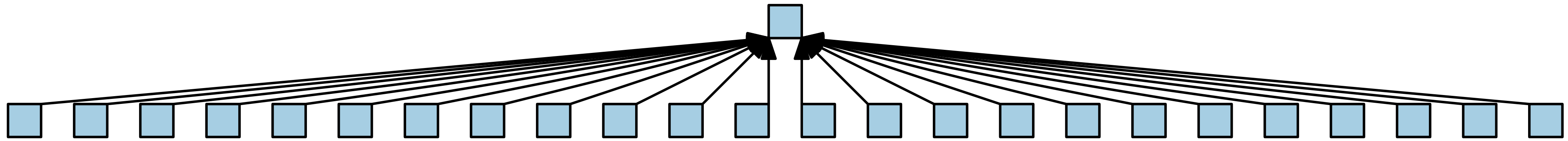
Can be formulated as an integer linear program:

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

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.

- 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.

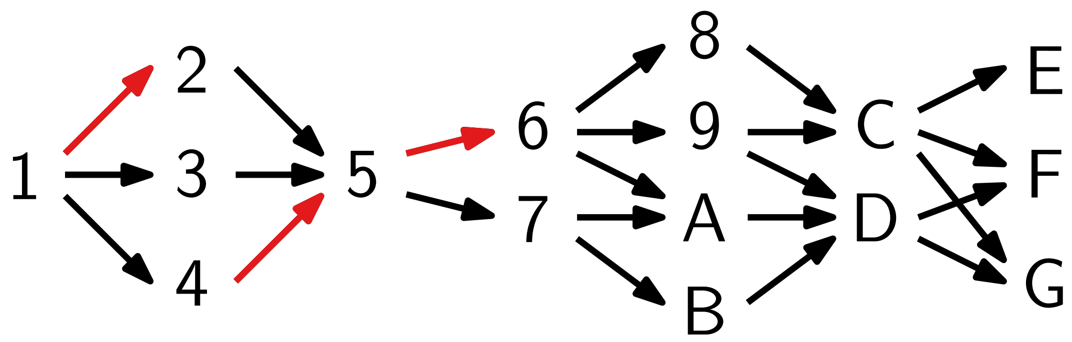
- Input: n jobs with unit processing time, W identical machines, partial ordering $<$ on the jobs.
- Output: Schedule respecting $<$ such that completion time (known as *makespan*) is minimized.
- NP-hard, $\boxed{(2 - 1/W)\text{-approximation}}$, no $(4/3 - \varepsilon)$-approximation ($W \geq 3$)

Approximating Precedence-Const. Multi-Processor Scheduling

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



Number of machines is $W = 2$.

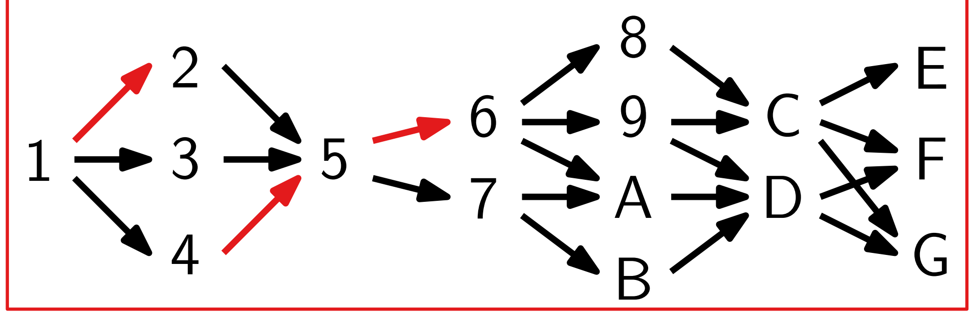
Output: Schedule

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“

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

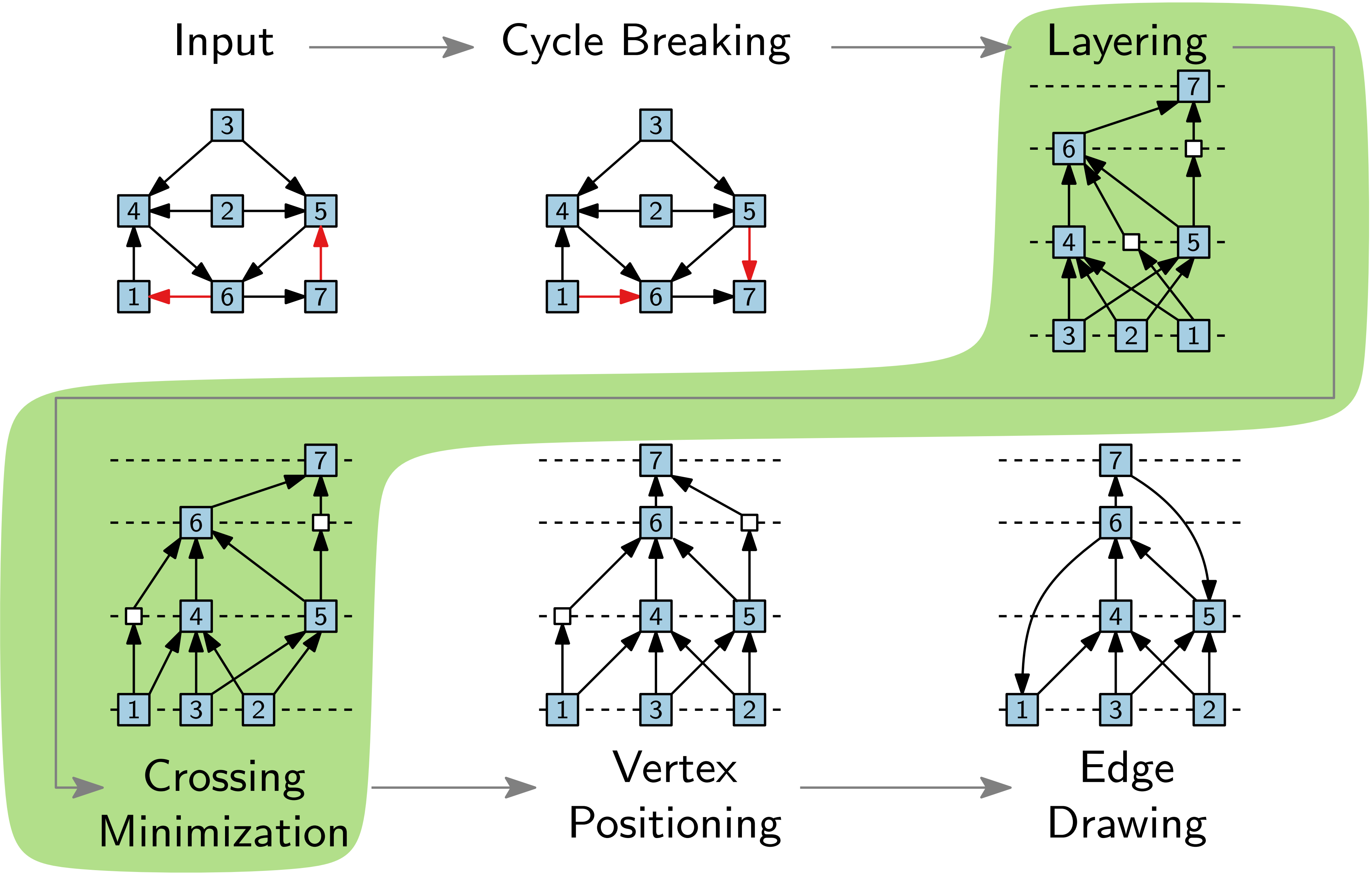
Goal: measure the quality of our algorithm using the lower bounds

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

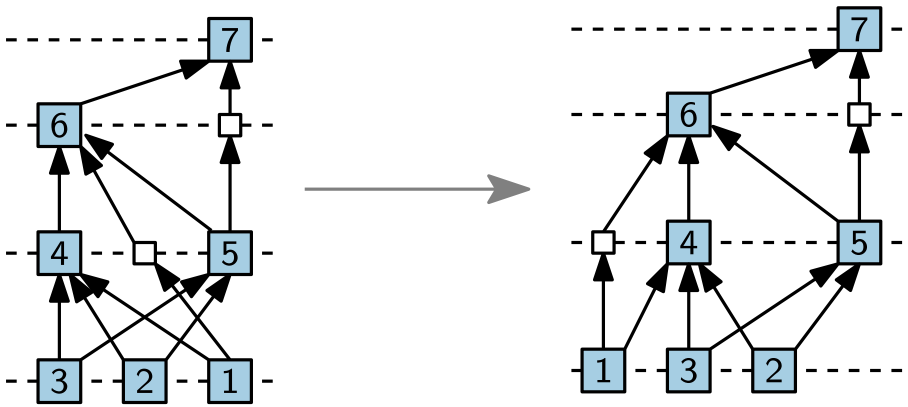
Bound. ALG $\leq \left\lceil \frac{n+\ell}{2} \right\rceil \approx \lceil n/2 \rceil + \ell/2 \leq 3/2 \cdot$ OPT

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

Step 3: Crossing Minimization



Step 3: Crossing Minimization



Problem.

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

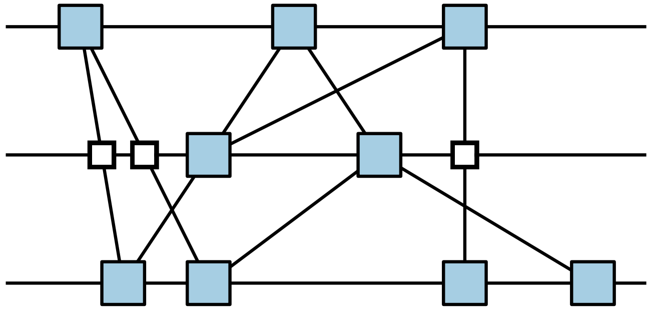
- NP-hard, even for two layers. [Garey & Johnson '83]
- Hardly any approaches optimize over multiple layers. ☹

Iterative Crossing Reduction

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

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

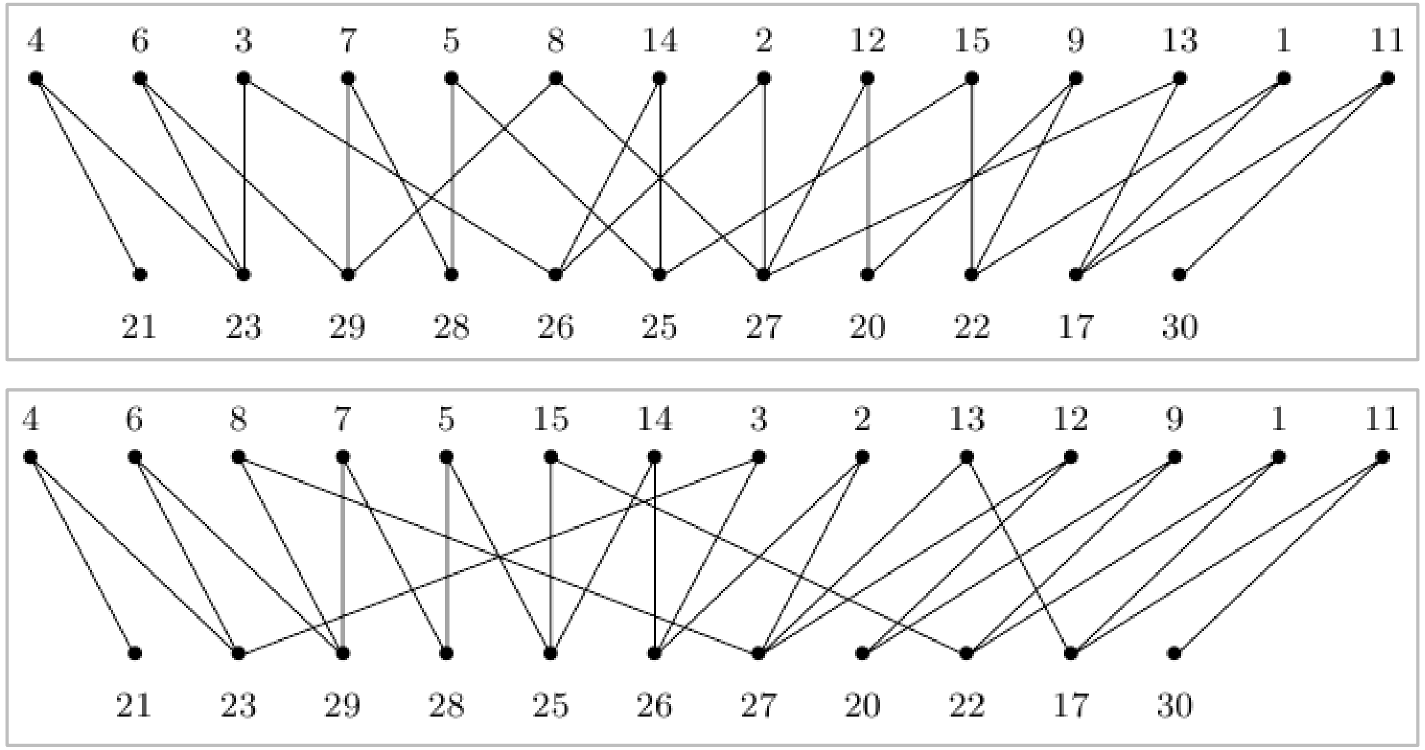
Problem.

- 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]

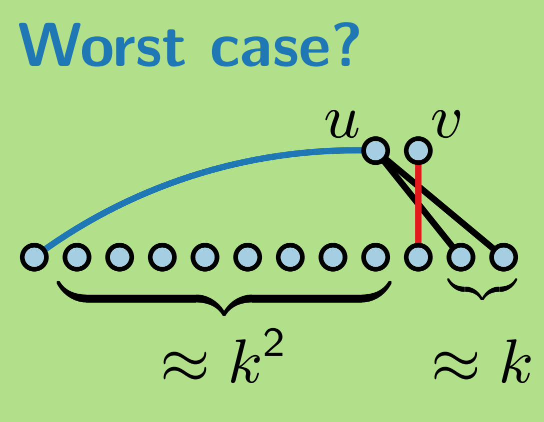
Barycenter Heuristic

[Sugiyama et al. '81]

- Intuition: There are few crossing if vertices are "close" to their neighbors.

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

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



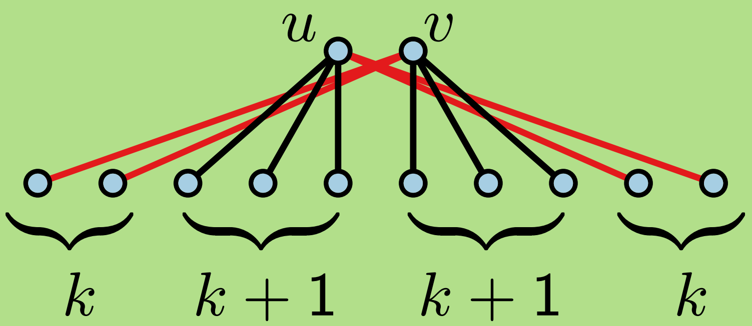
- To get π_2, sort L_2 ascendingly according to bary$(\cdot)$.
- Vertices with the same barycenter keep their old relative ranks.
- Linear runtime (in the number of vertices and edges).
- Relatively good results in practice.
- Finds crossing-free solutions if they exist. ← Exercise!
- Factor-$O(\sqrt{|V(G)|})$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

Median Heuristic

[Eades & Wormald '94]

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



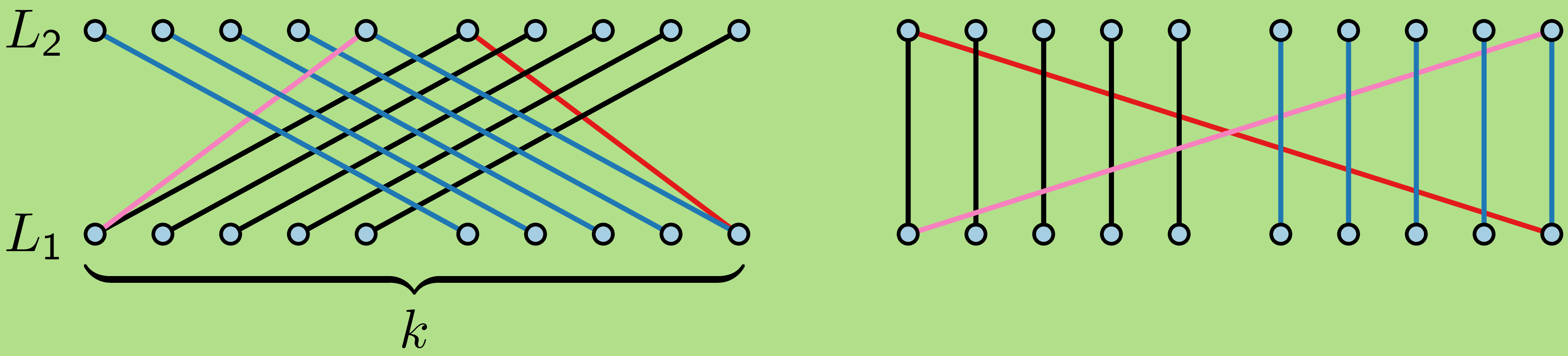
\# crossings: $2k(k+1) + k^2$ vs. $(k+1)^2$

Greedy-Switch Heuristic

[Eades & Kelly '86]

- Iteratively swap pairs of neighboring vertices on L_2 as long as the number of crossings decreases.
- 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: $\approx k^2/4$ $\approx 2k$

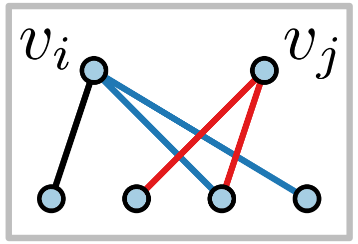
Integer Linear Program (ILP)

[Jünger & Mutzel, '97]

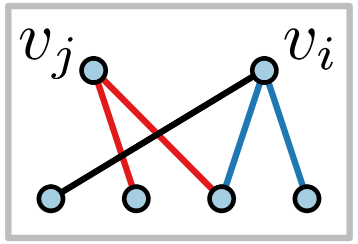
- Constant $c_{ij} :=$ # crossings between edges incident to v_i and v_j if $\pi_2(v_i) < \pi_2(v_j)$

- 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:

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

Integer Linear Program (ILP)

- Objective (minimize the number of crossings):

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

- Transitivity constraints:

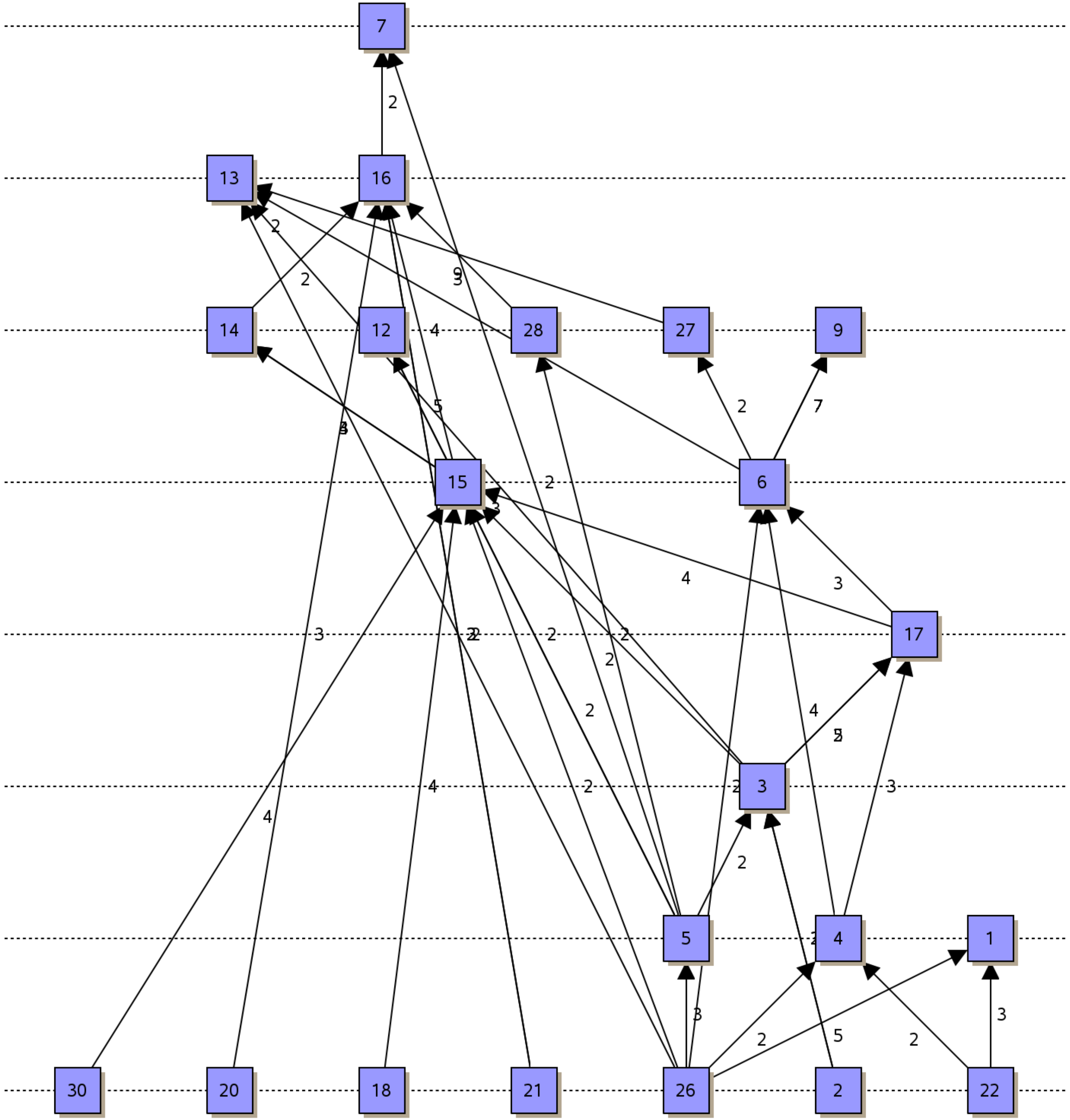
$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n_2$$

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

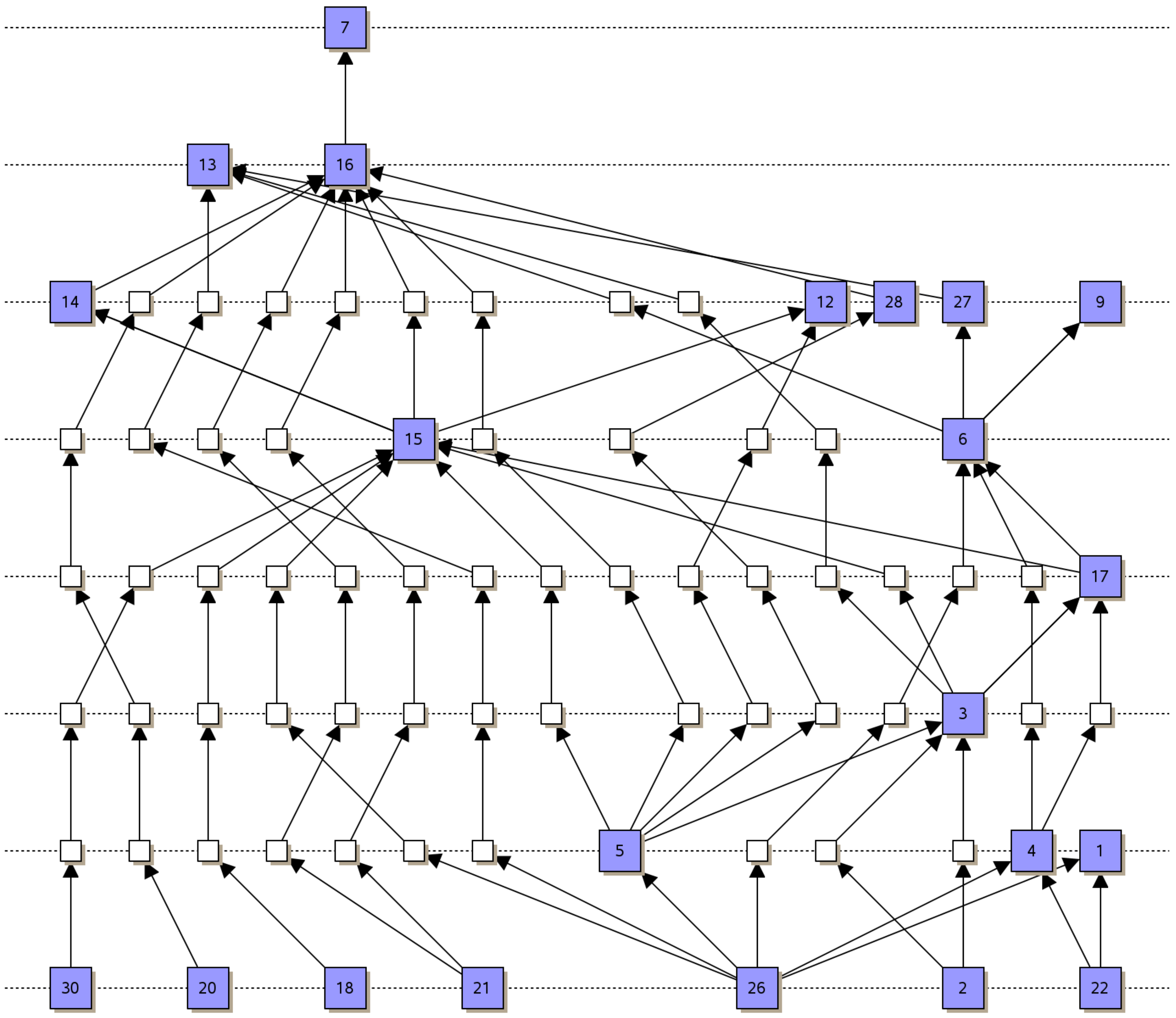
Properties.

- 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

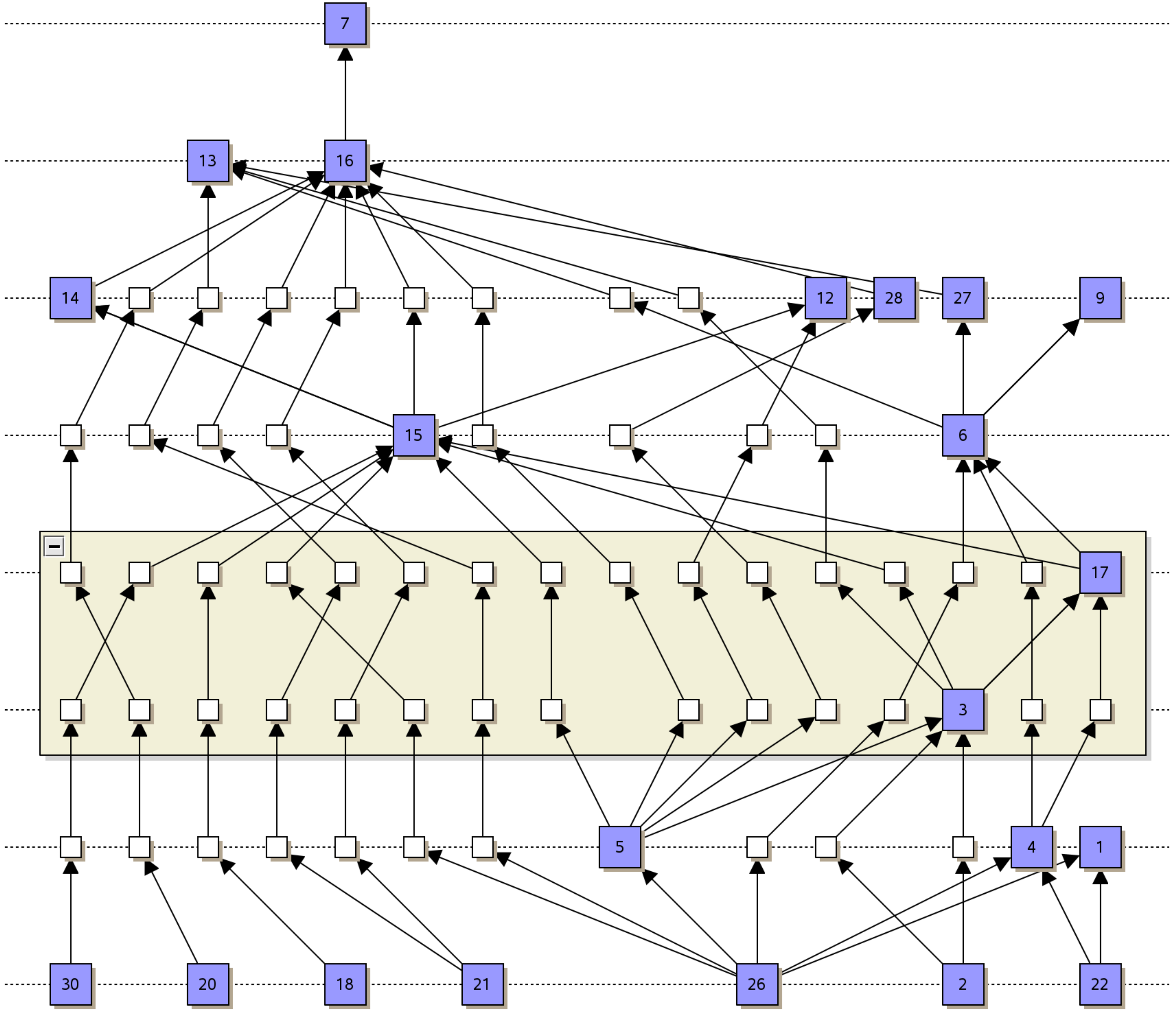
Iterations on Example



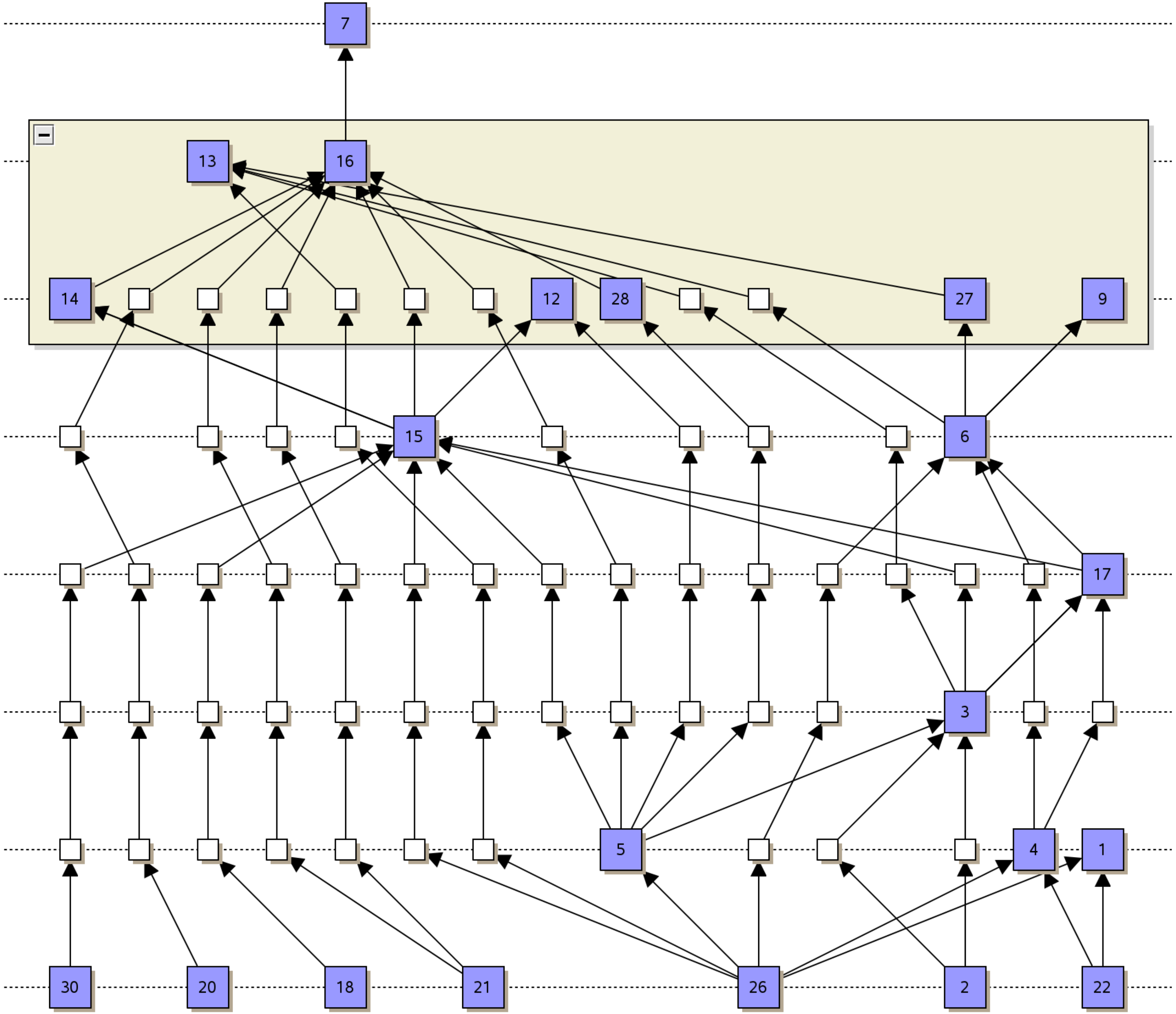
Iterations on Example



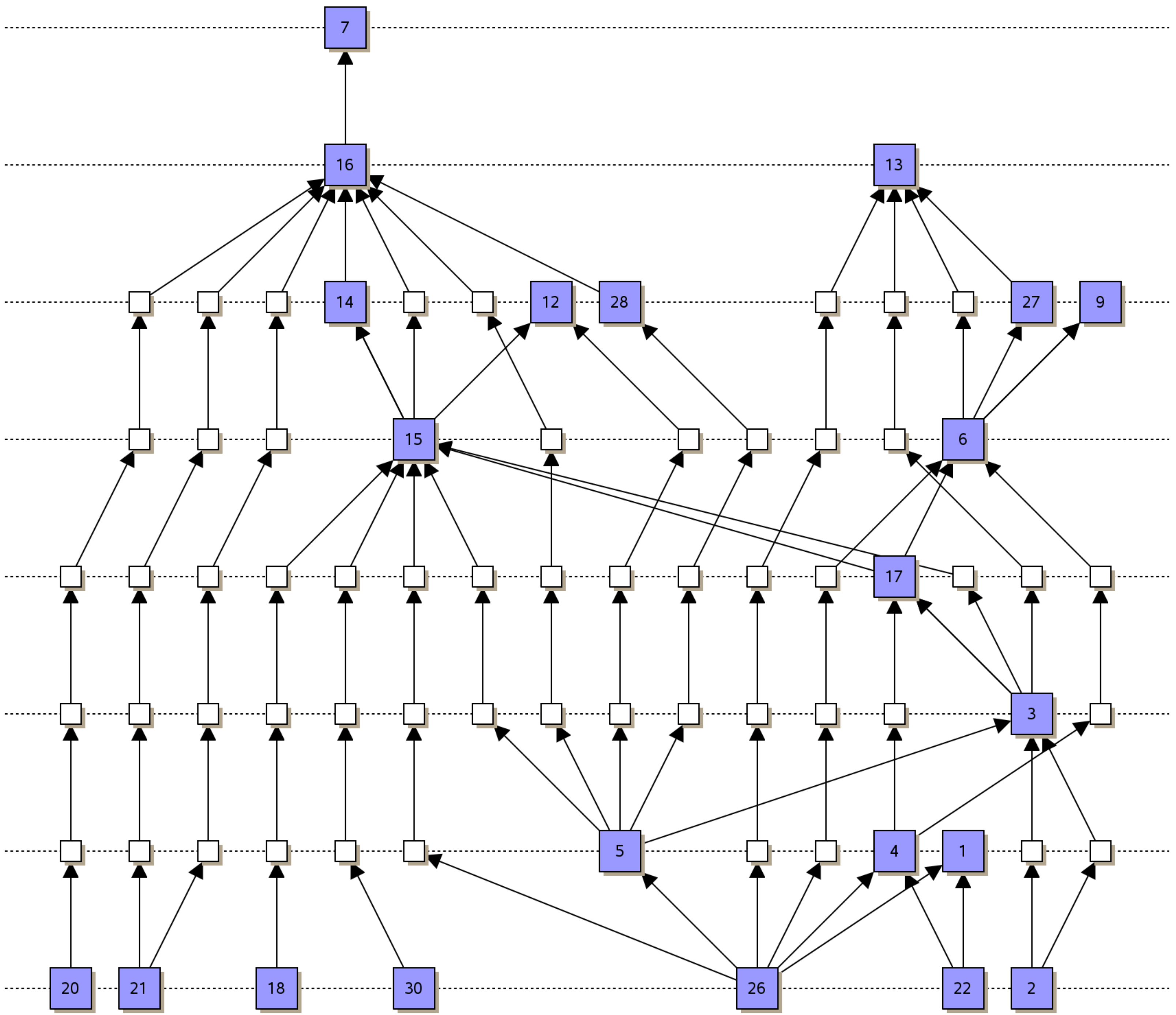
Iterations on Example



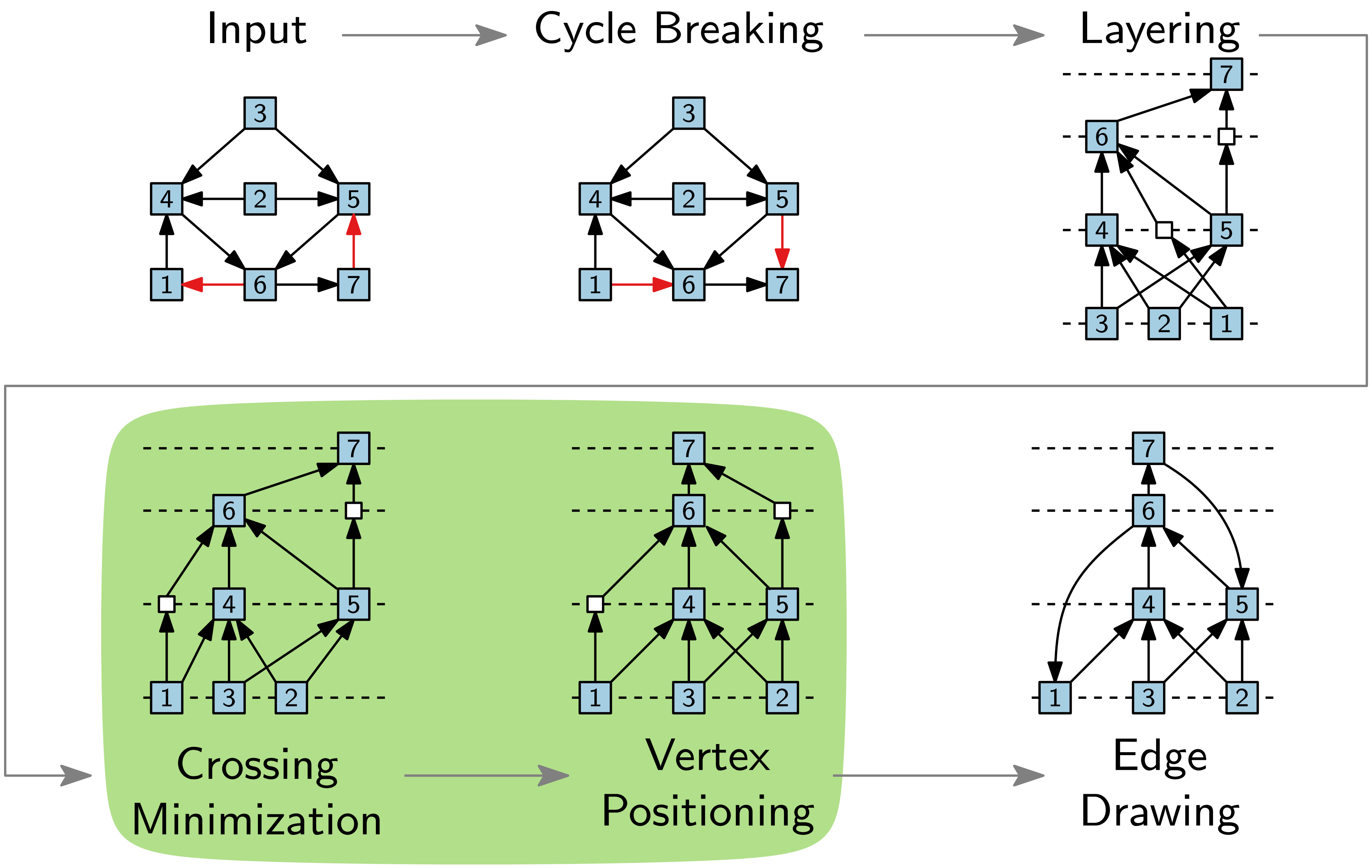
Iterations on Example



Iterations on Example



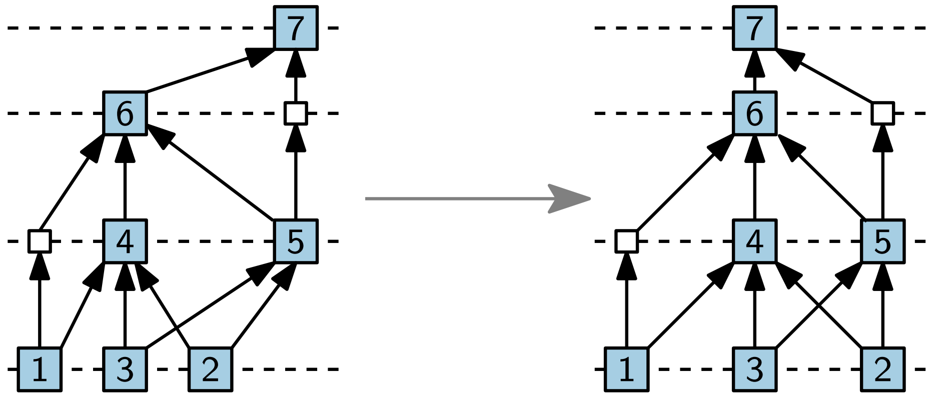
Step 4: Vertex Positioning



Step 4: Vertex Positioning

Goals.

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



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

Quadratic Program

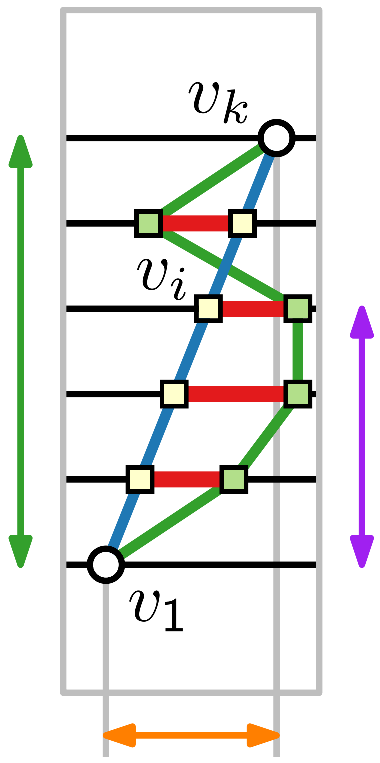
- Let $e = (v_1, v_k)$ be an edge of G, and let $p_e = (v_1, \dots, v_k)$ be the corresponding path with dummy vertices $v_2, \dots, v_{k-1}$.
- x-coordinate of v_i according to the line segment $\overline{v_1 v_k}$ (with equal spacing of the layers):

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

- 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:
  $x(w) - x(v) \geq \rho$ ← min. horizontal distance



- QP is time-expensive.
- Width can be exponential.

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)

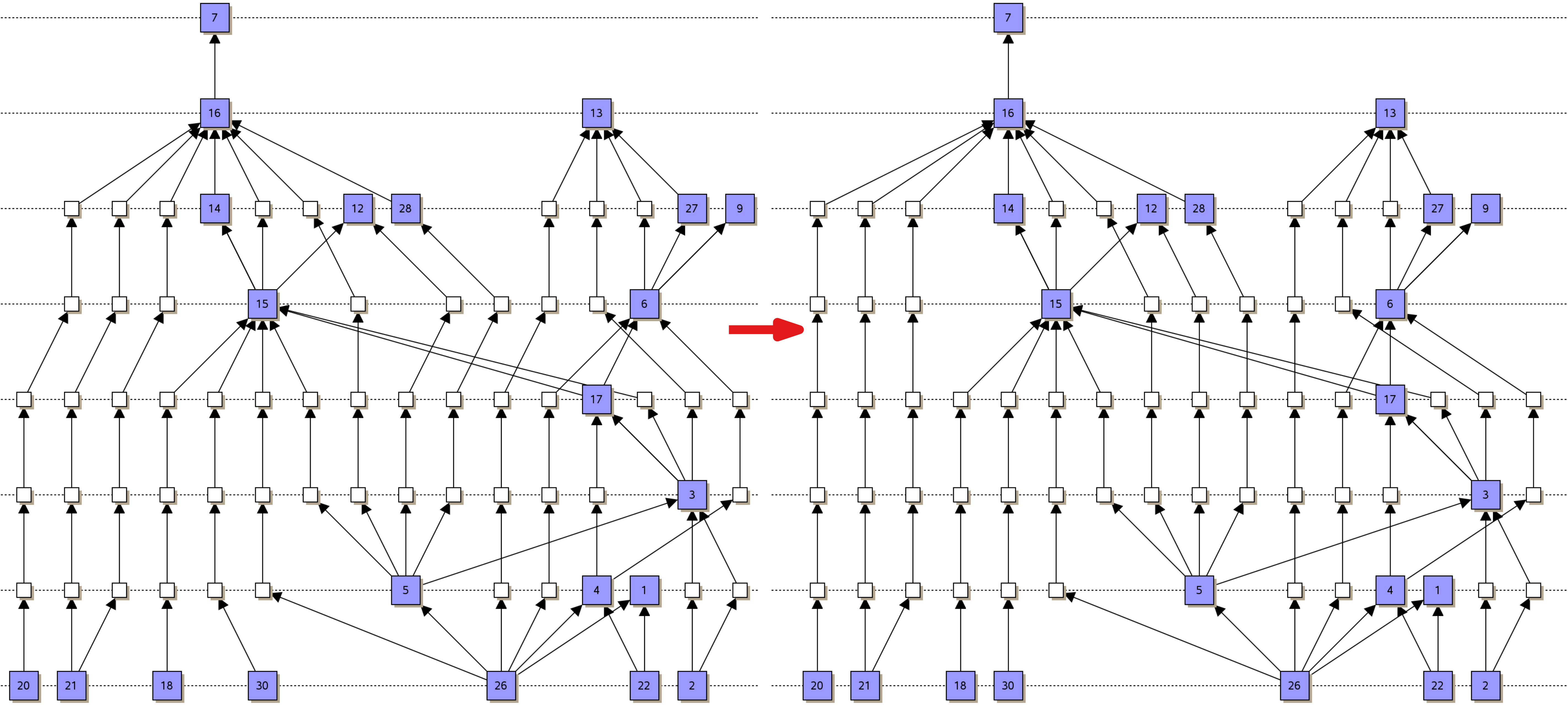
- Other algorithms, e.g., the one of Brandes and Köpf [GD 2002, see also Brandes, Walter, Zink: arXiv 2020]:

  – tries to align vertices vertically

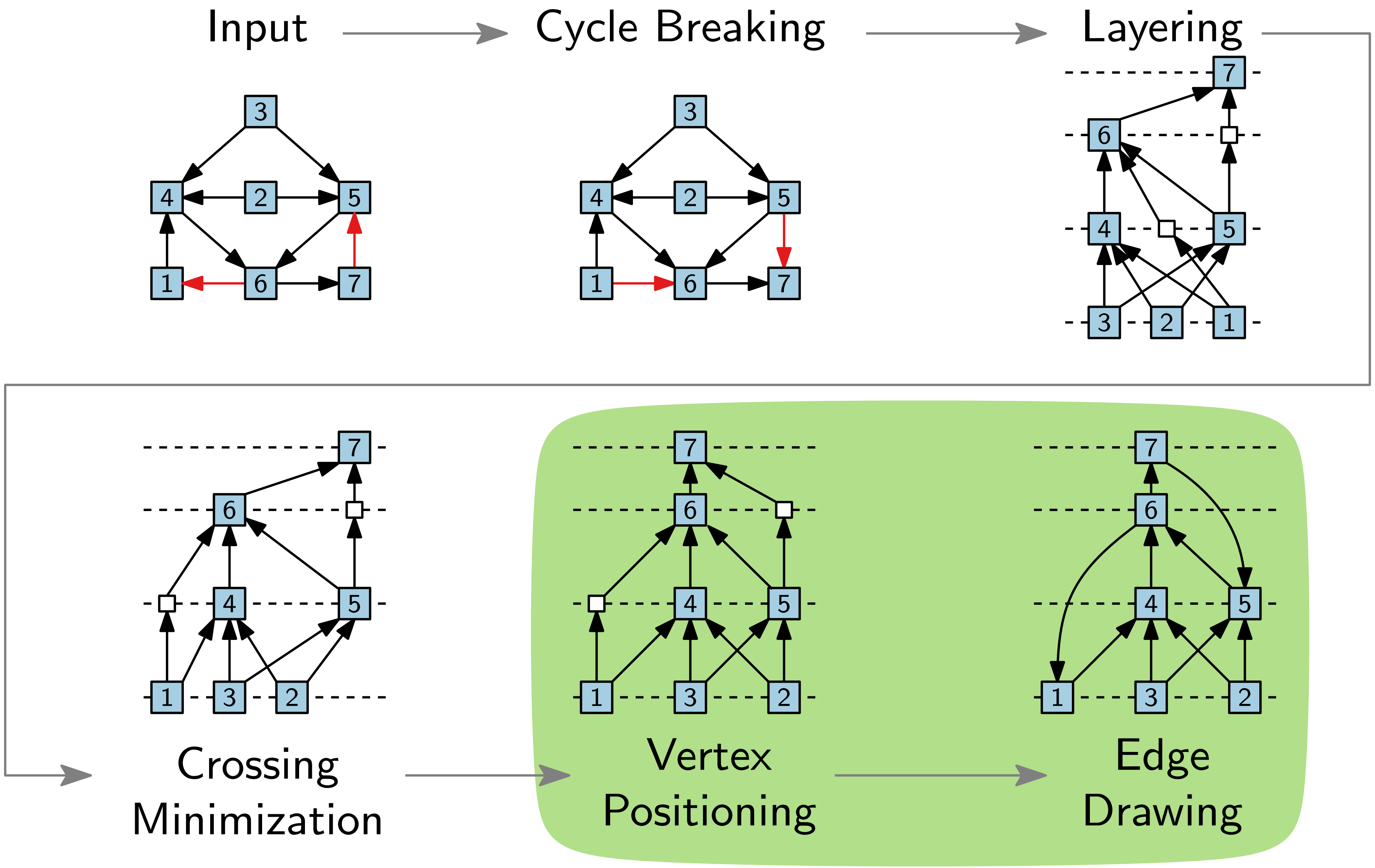
  – does horizontal compaction afterwards

  – linear running time

Example

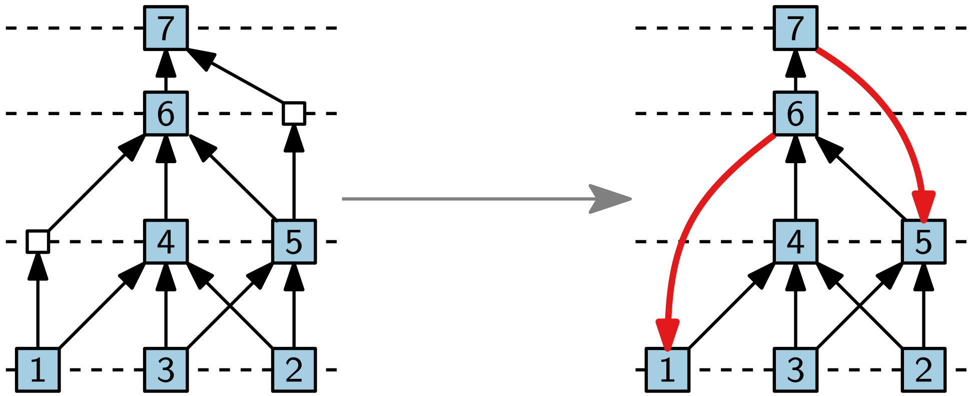

7
16
13
14
12
28
27
9
15
6
17
3
5
4
1
20
21
18
30
26
22
2
7
16
13
14
12
28
27
9
15
6
17
3
5
4
1
20
21
18
30
26
22
2

Step 5: Drawing Edges



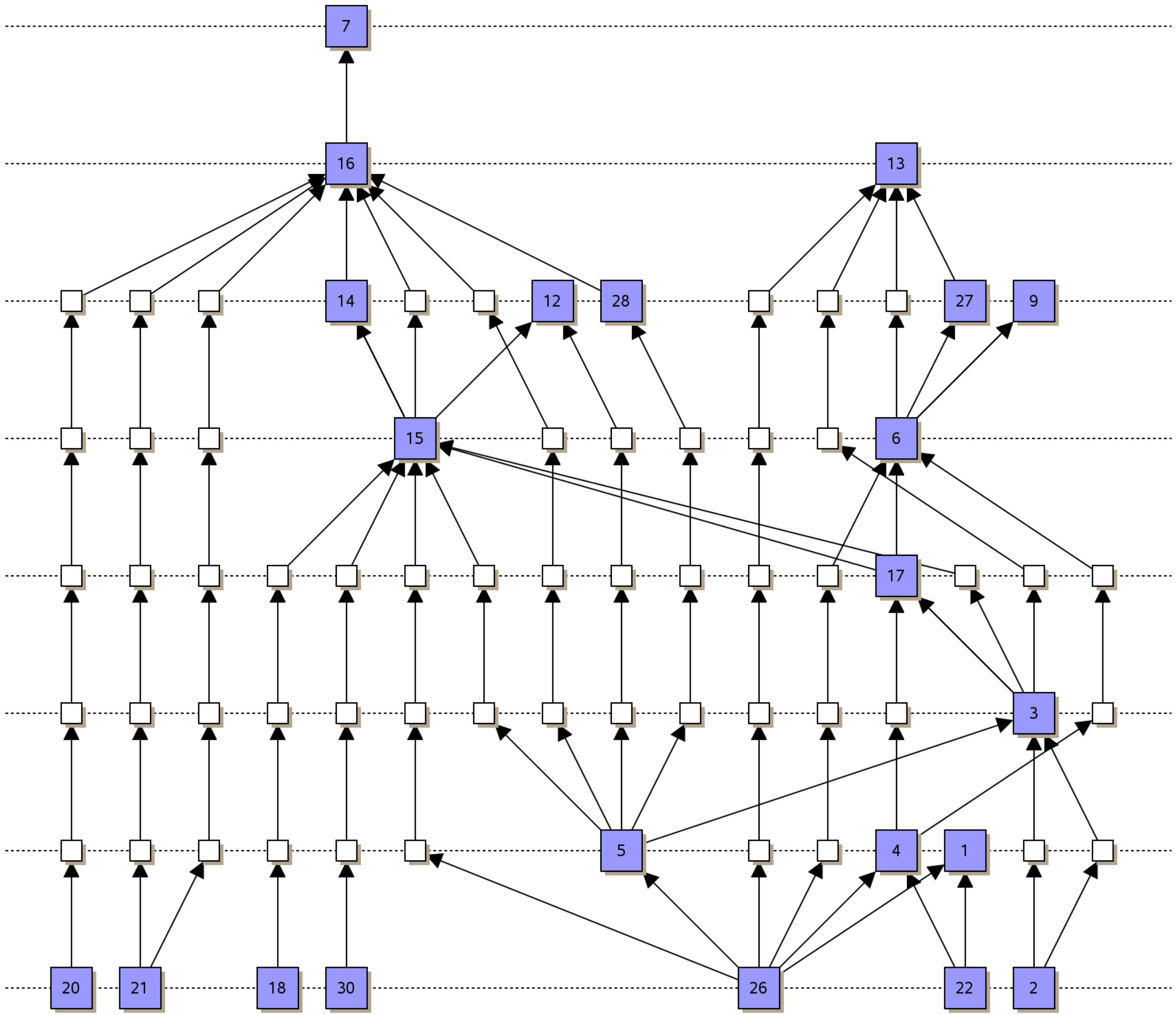
Step 5: Drawing Edges

Possibility.
Substitute polylines by Bézier curves.

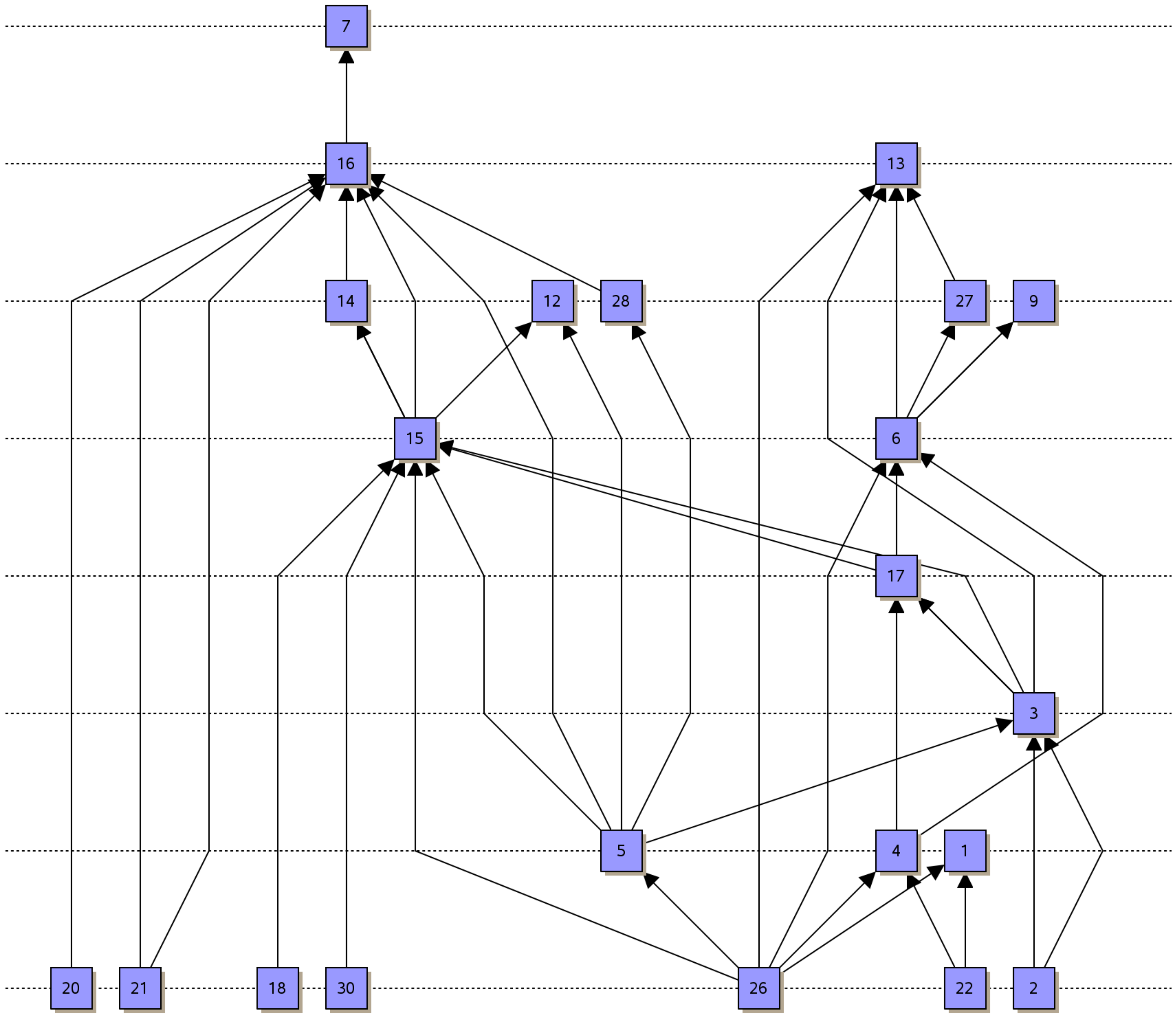


Remark.
Draw reversed edges downwards.

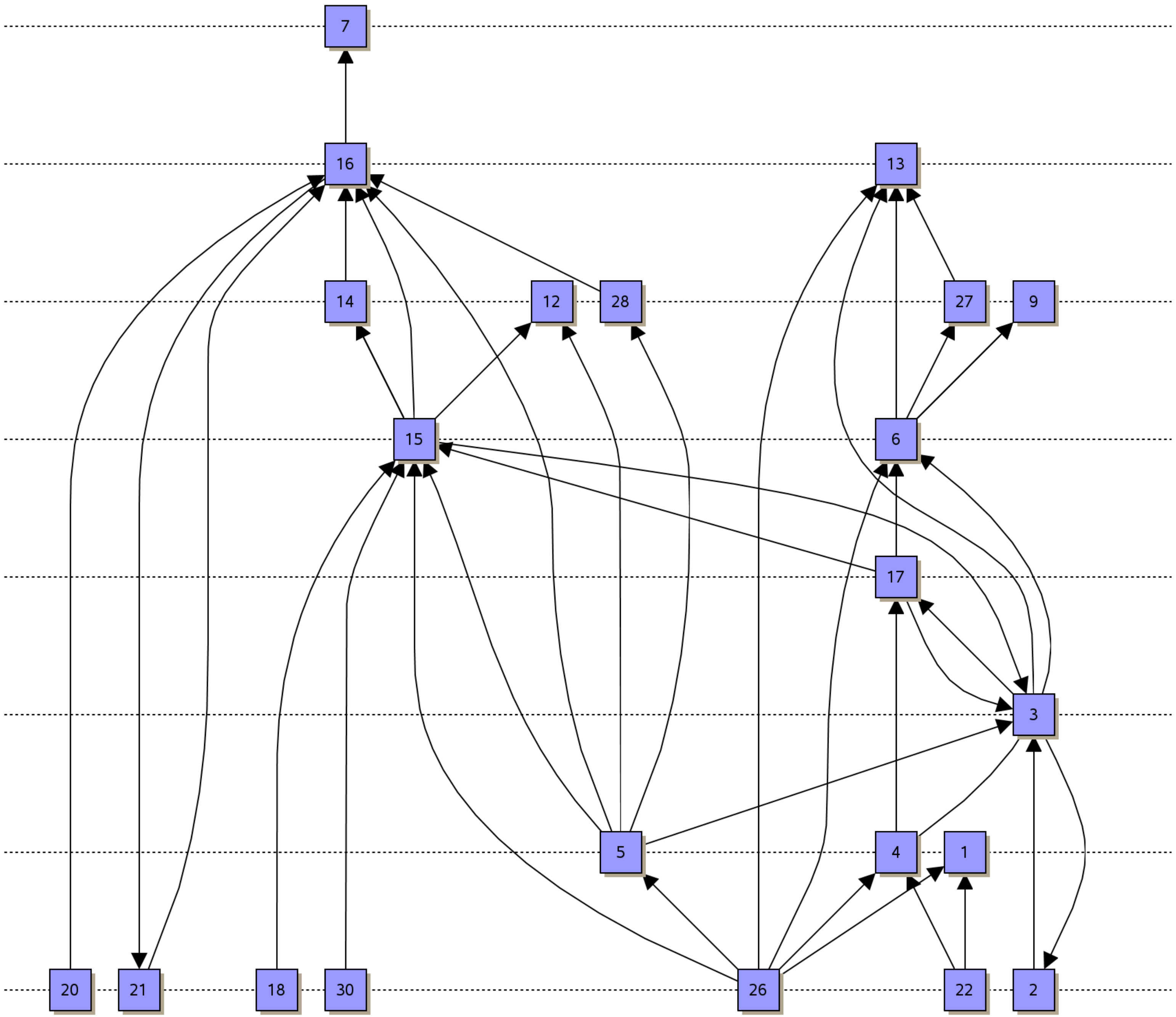
Example



Example

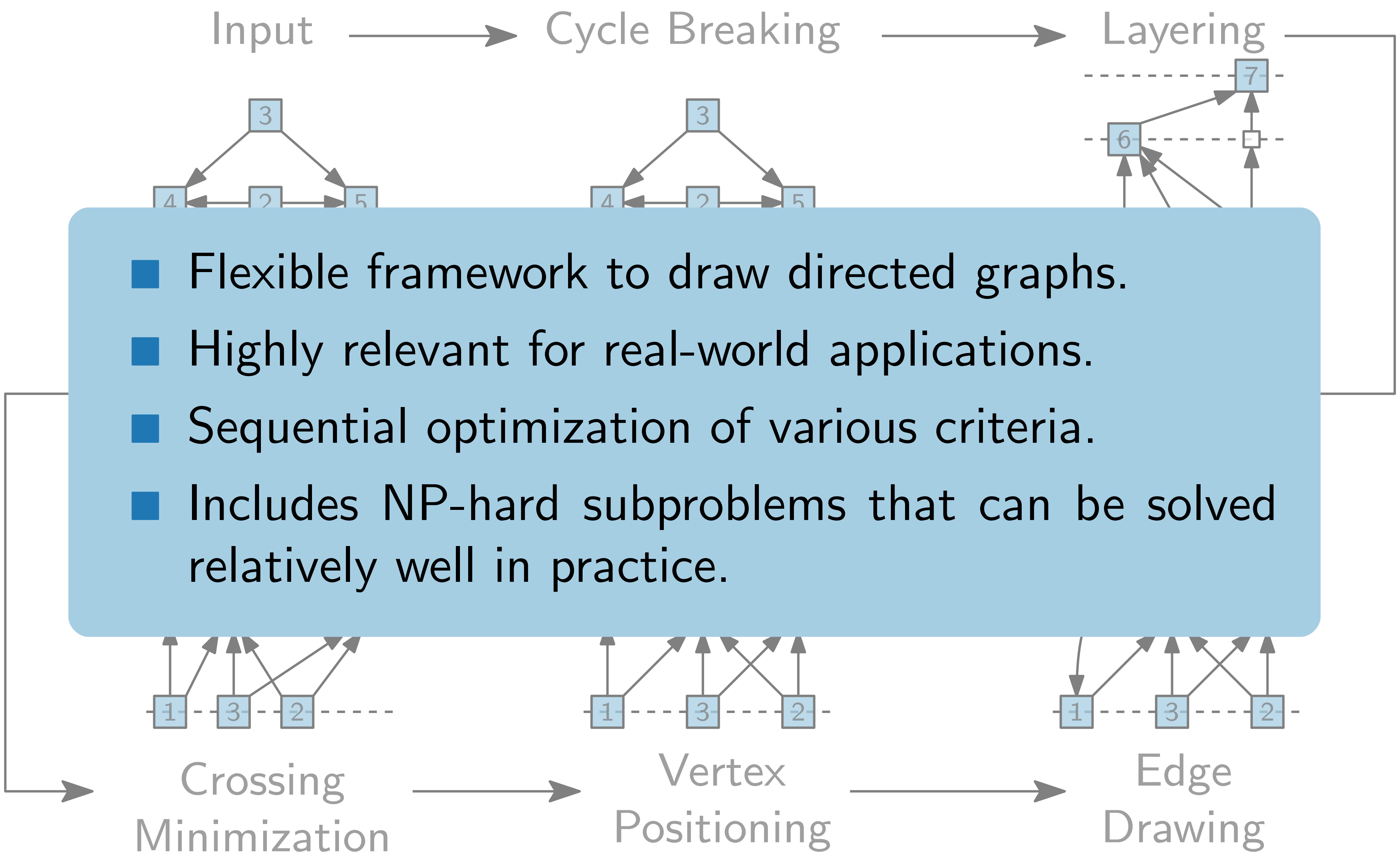


Example



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