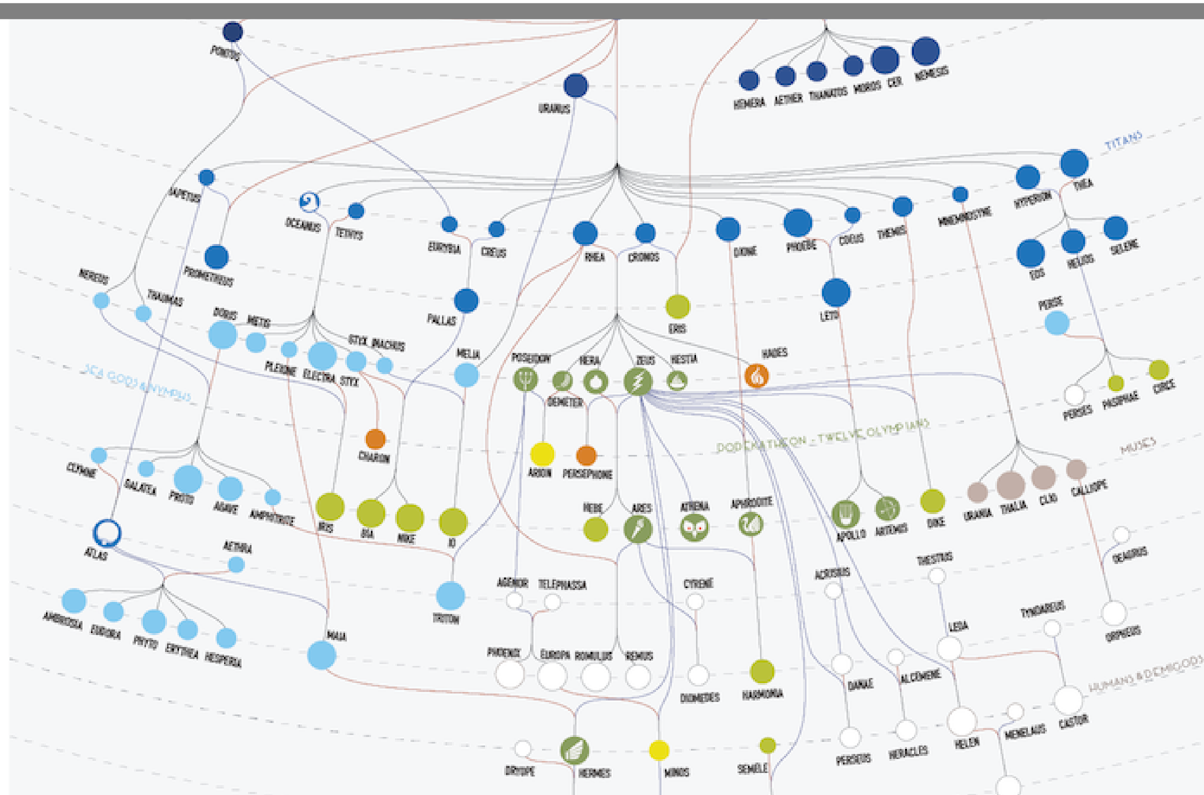
Algorithm for
Visualization of Directed Graphs

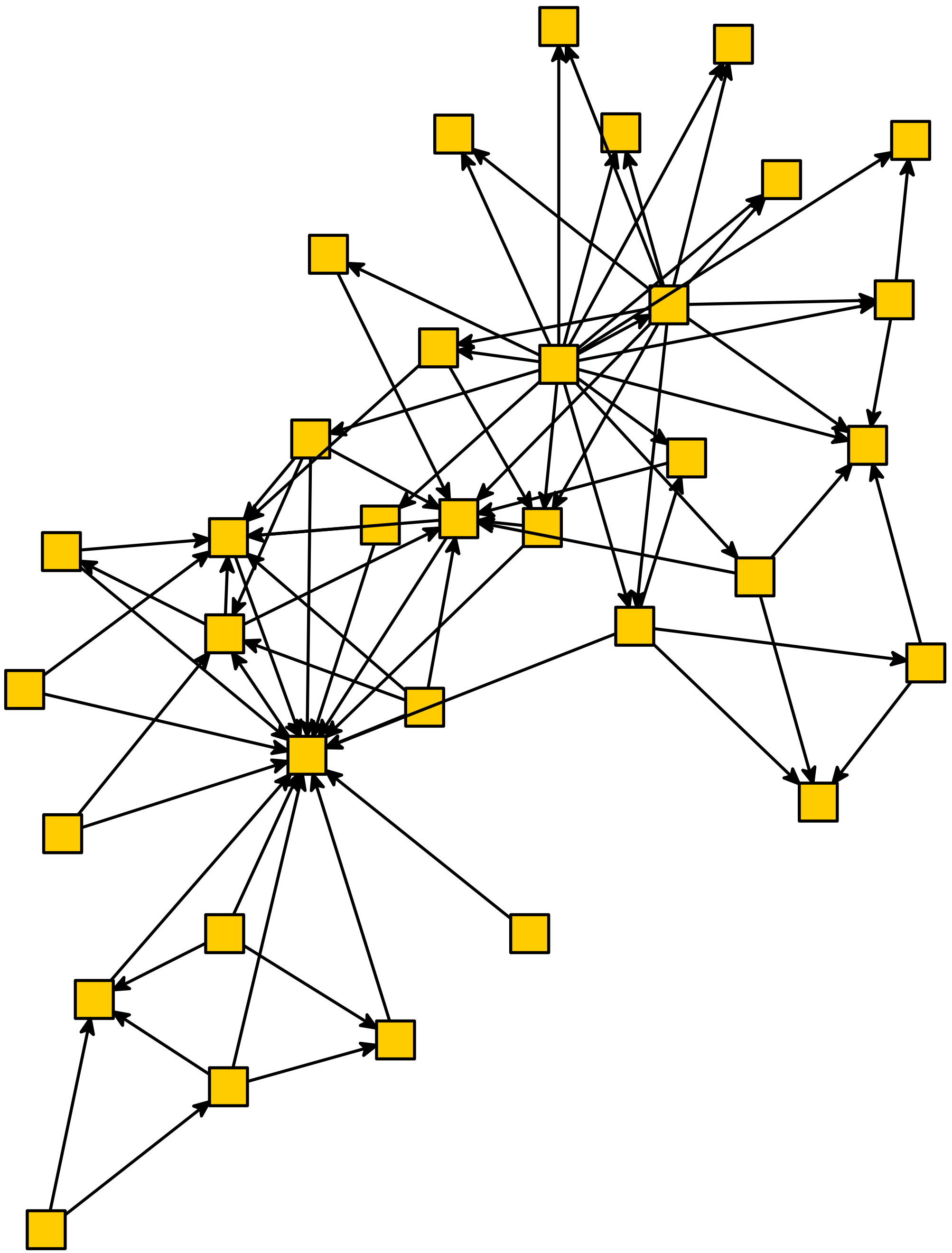
Course : Data Visualization
Lecturer : Tamara Mchedlidze
Utrecht University, Dept. of Information and Computing Sciences



Lecture Overview

- How to draw a directed graph
- Examples
- Sugiyama framework

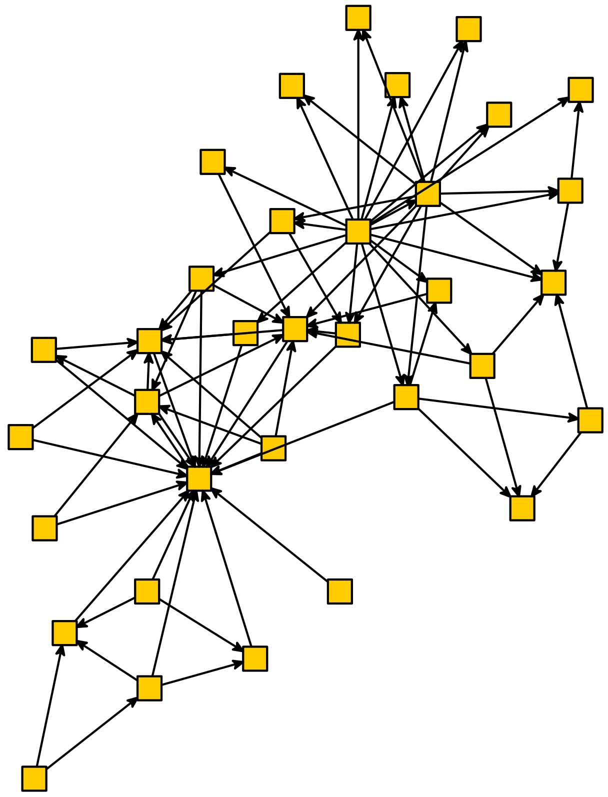
Example

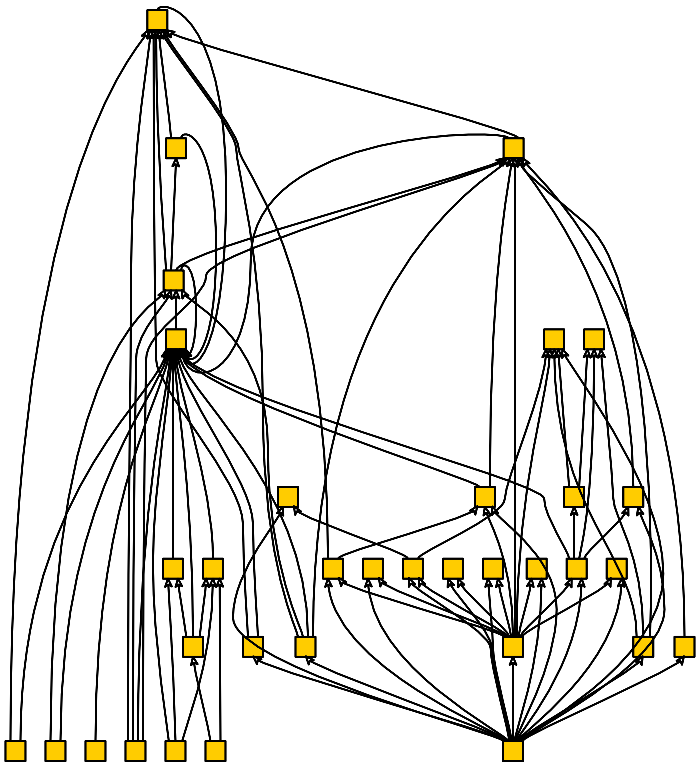


Layered Layout

Given: directed graph $D = (V, A)$

Find: drawing of D that emphasized the hierarchy

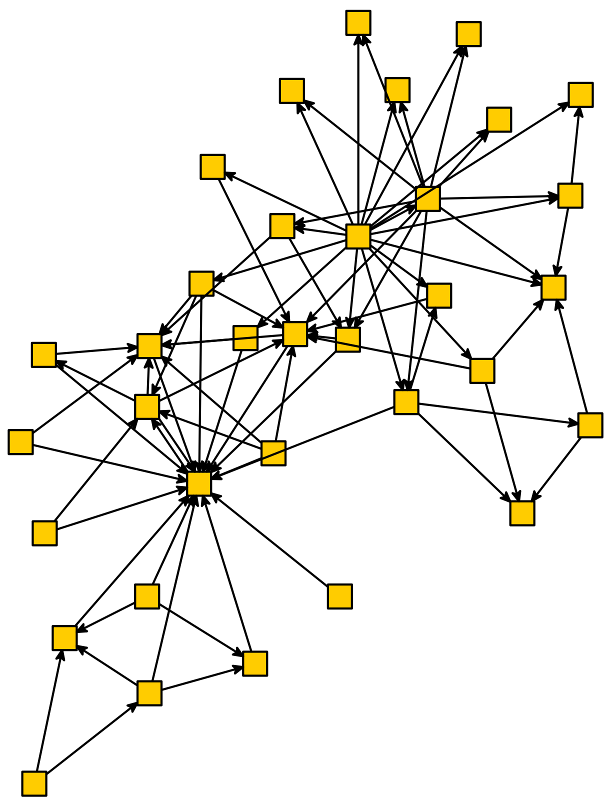


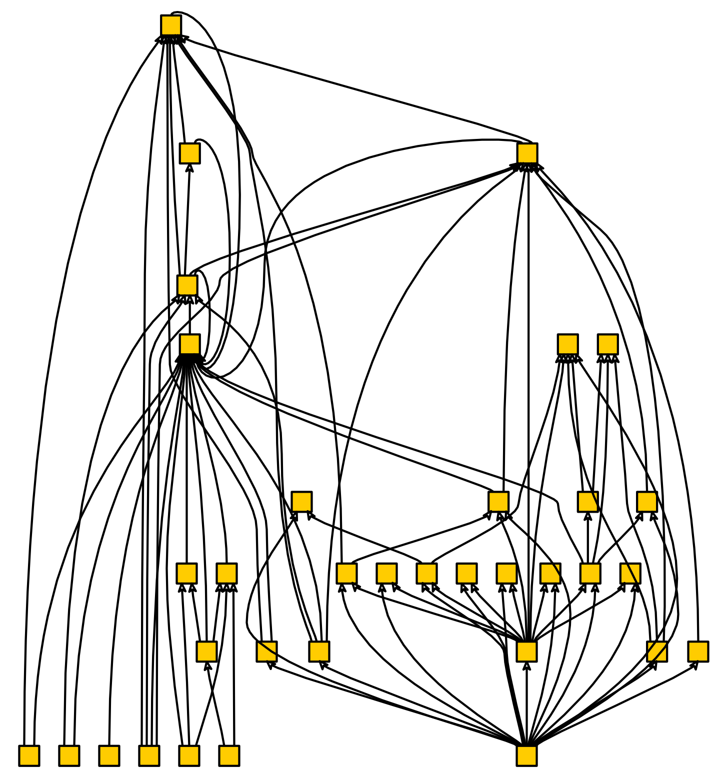


Layered Layout

Given: directed graph $D = (V, A)$

Find: drawing of D that emphasized the hierarchy



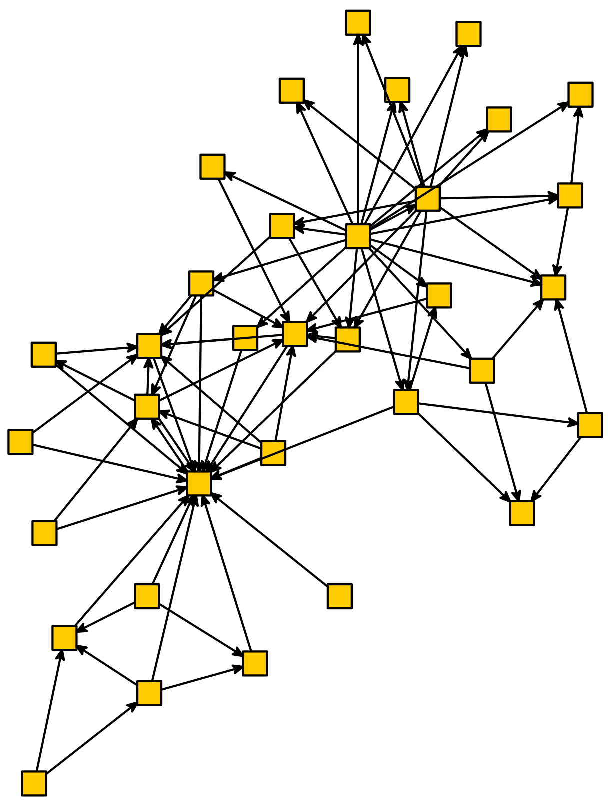


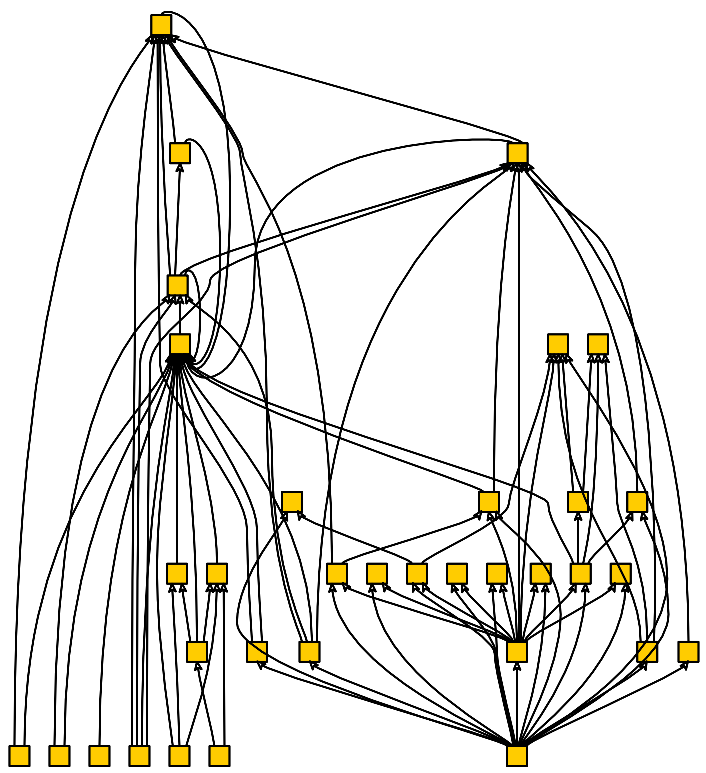
- many edges pointing to the same direction
- nodes lie on (few) horizontal lines

Layered Layout

Given: directed graph $D = (V, A)$

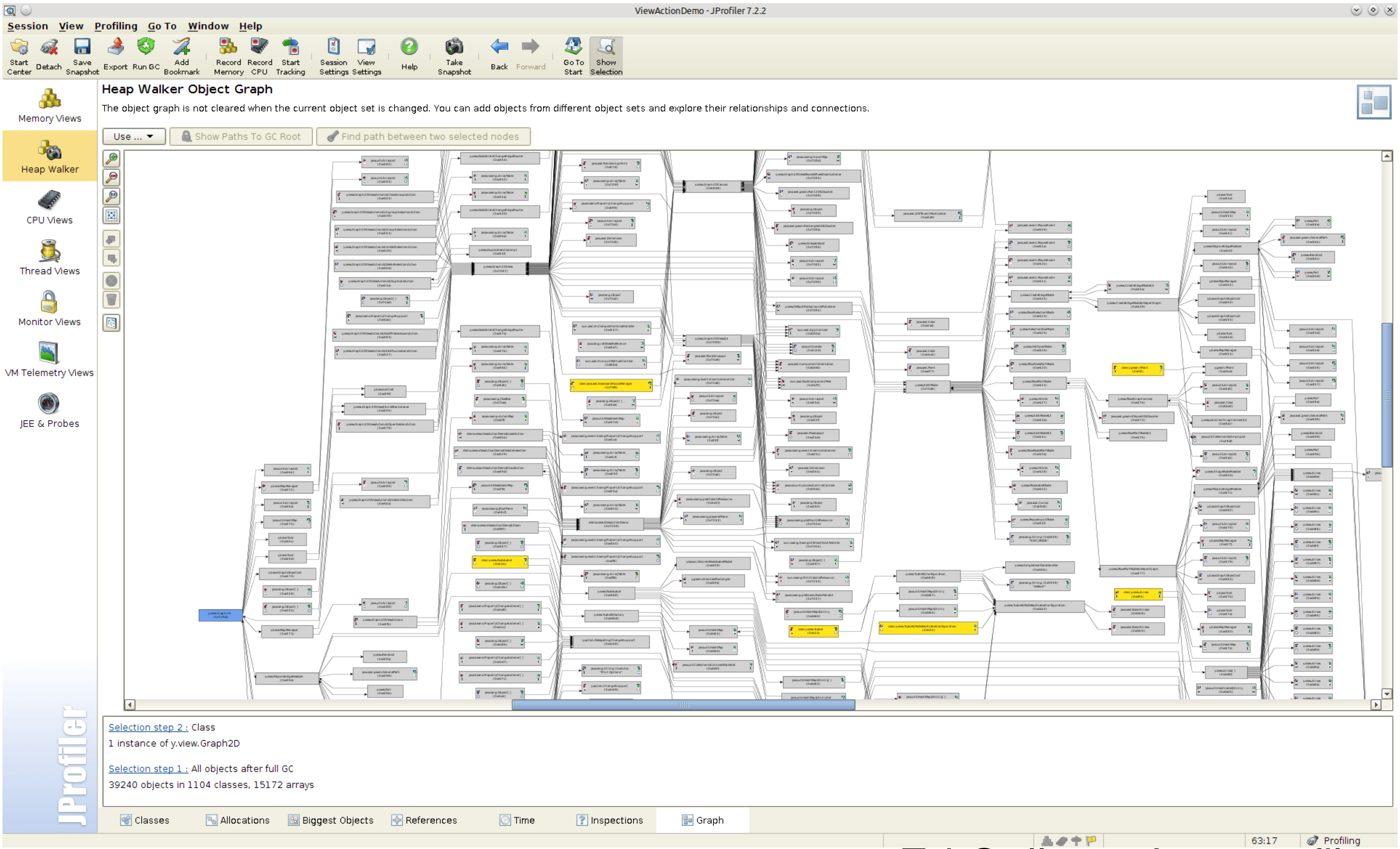
Find: drawing of D that emphasized the hierarchy





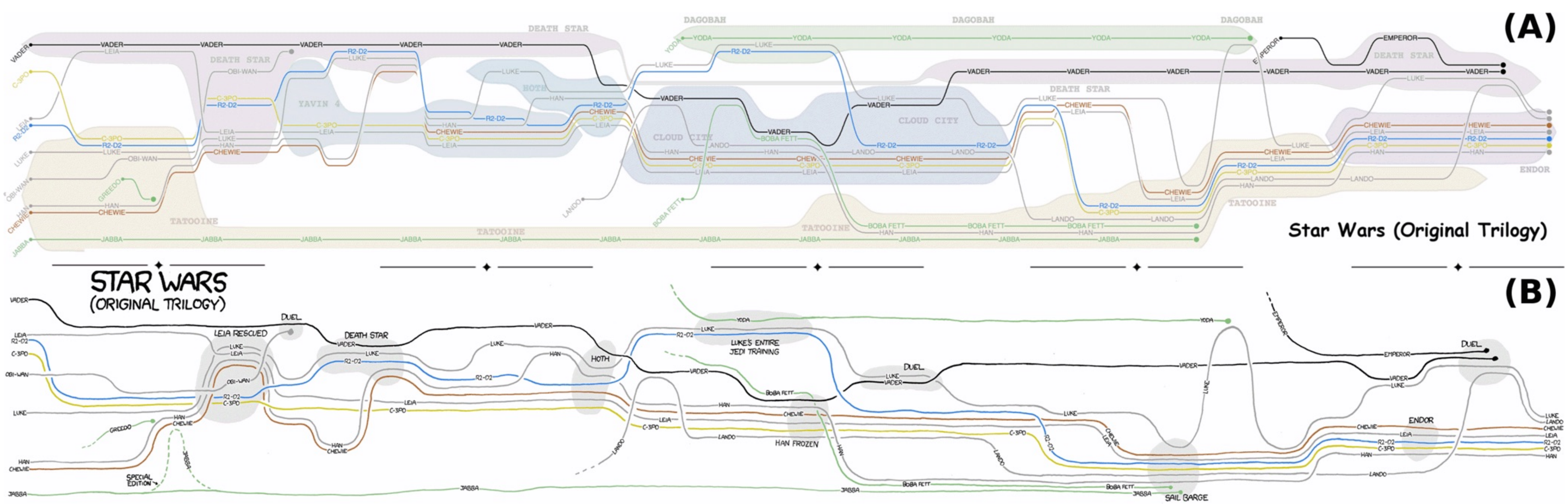
- edges as straight as possible and short
- few edge crossings
- nodes distributed evenly

Application: Java Profiler



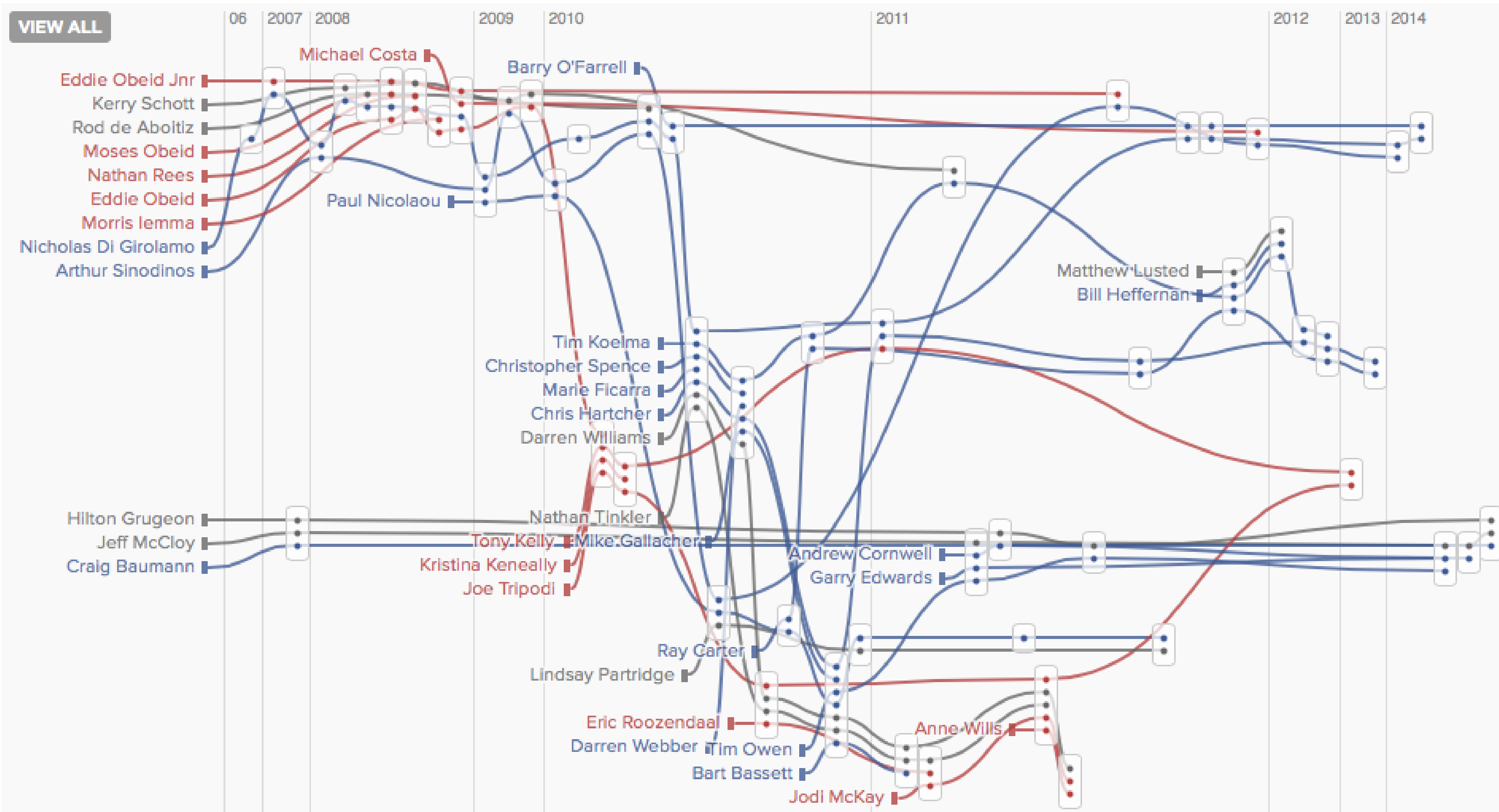
yEd Gallery: Java profiler
JProfiler using yFiles

Application: Storylines



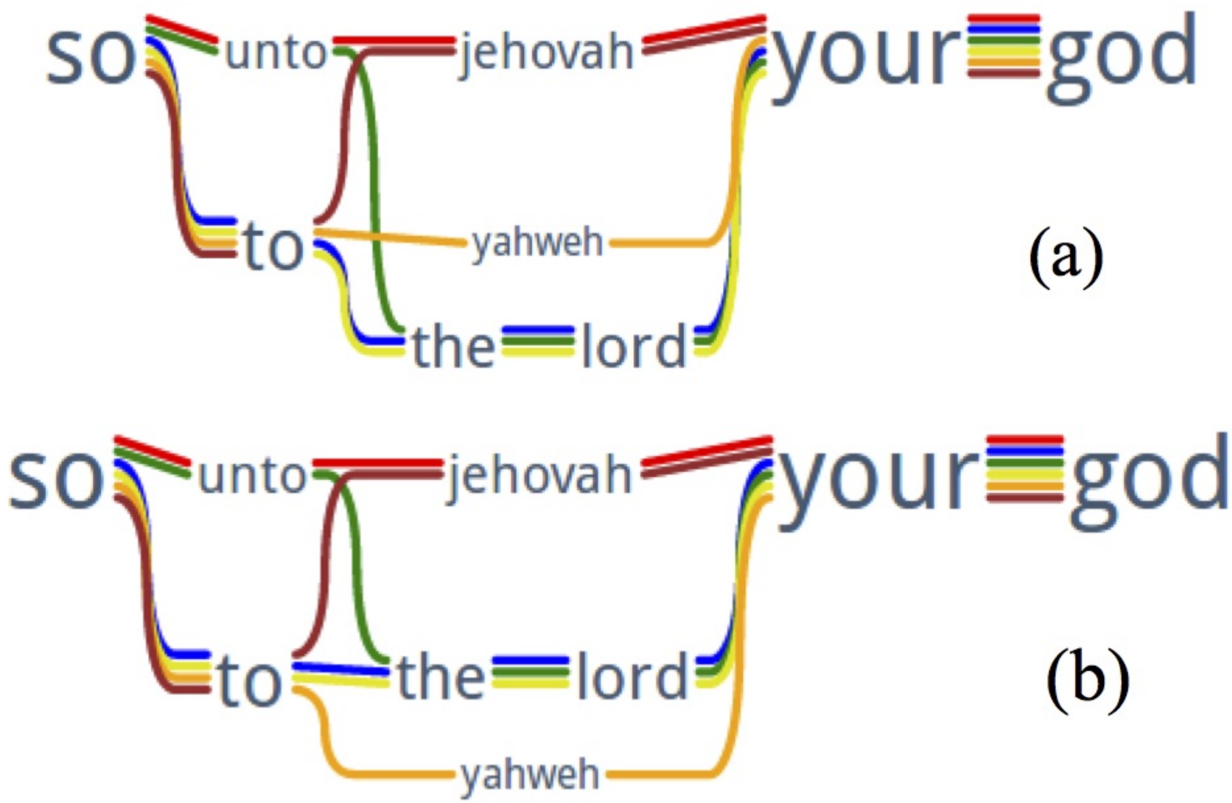
Source: "Design Considerations for Optimizing Storyline Visualizations" Tanahashi et al.

Application: Storylines



Source: ABC news, Australia

Application: Text-Variant graphs

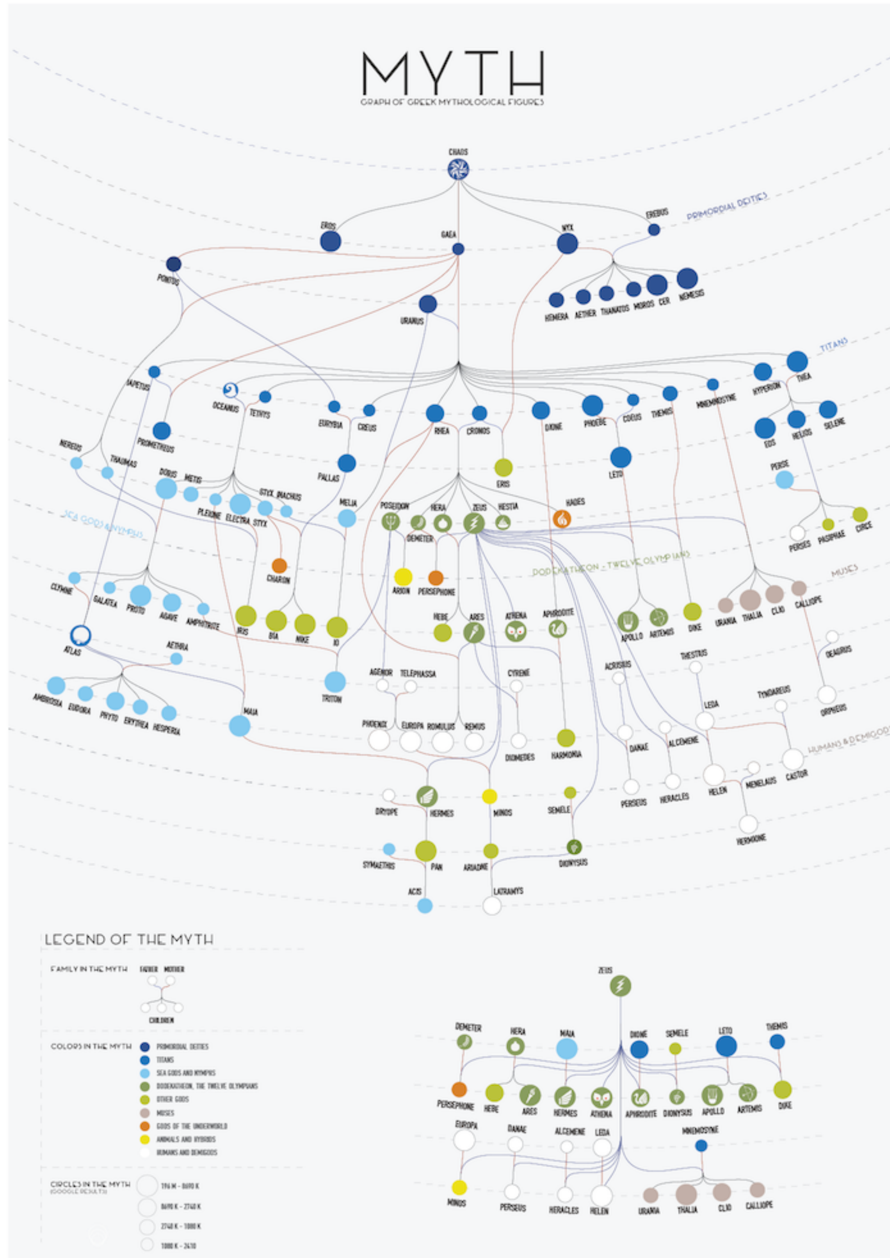


(a)

(b)

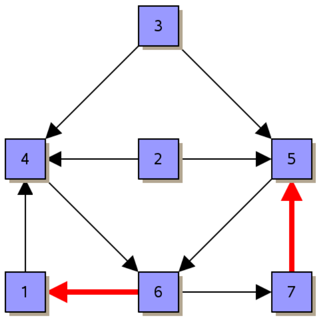
Source: Improving the Layout for Text Variant Graphs Jänicke et al.

Application: Mythological Creatures and Gods



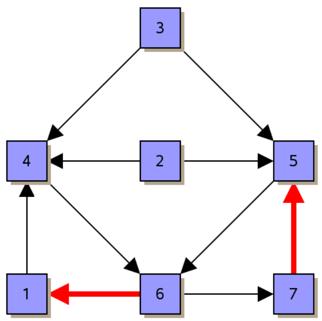
Source: Visualization that won the Graph Drawing contest 2016. Klawitter& Mchedlidze

Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)

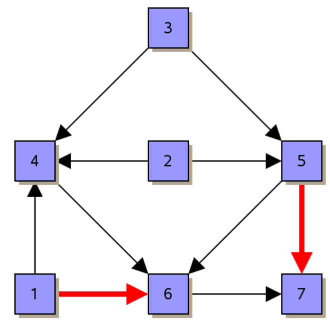


given

Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)

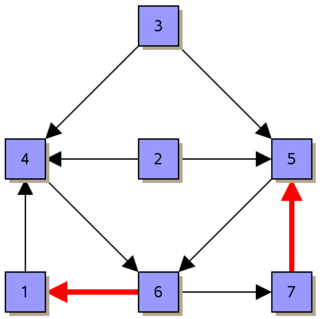


given

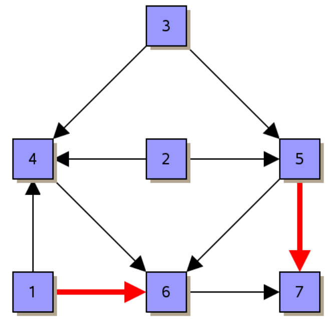


resolve cycles

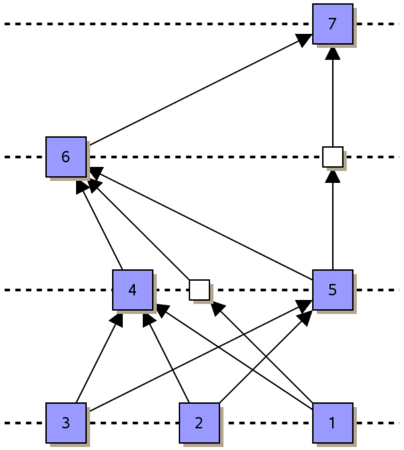
Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



given

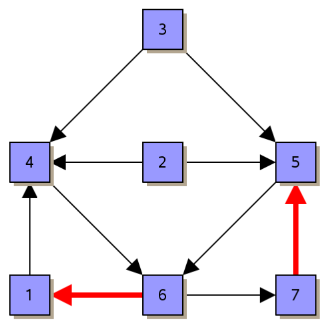


resolve cycles

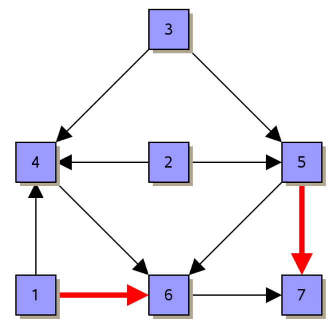


layer
assignemen

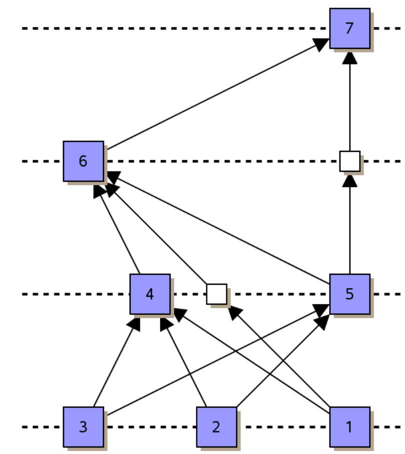
Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



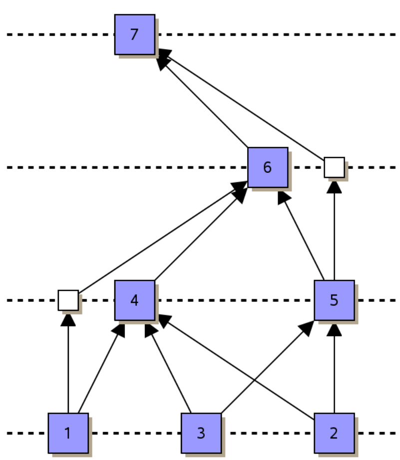
given



resolve cycles

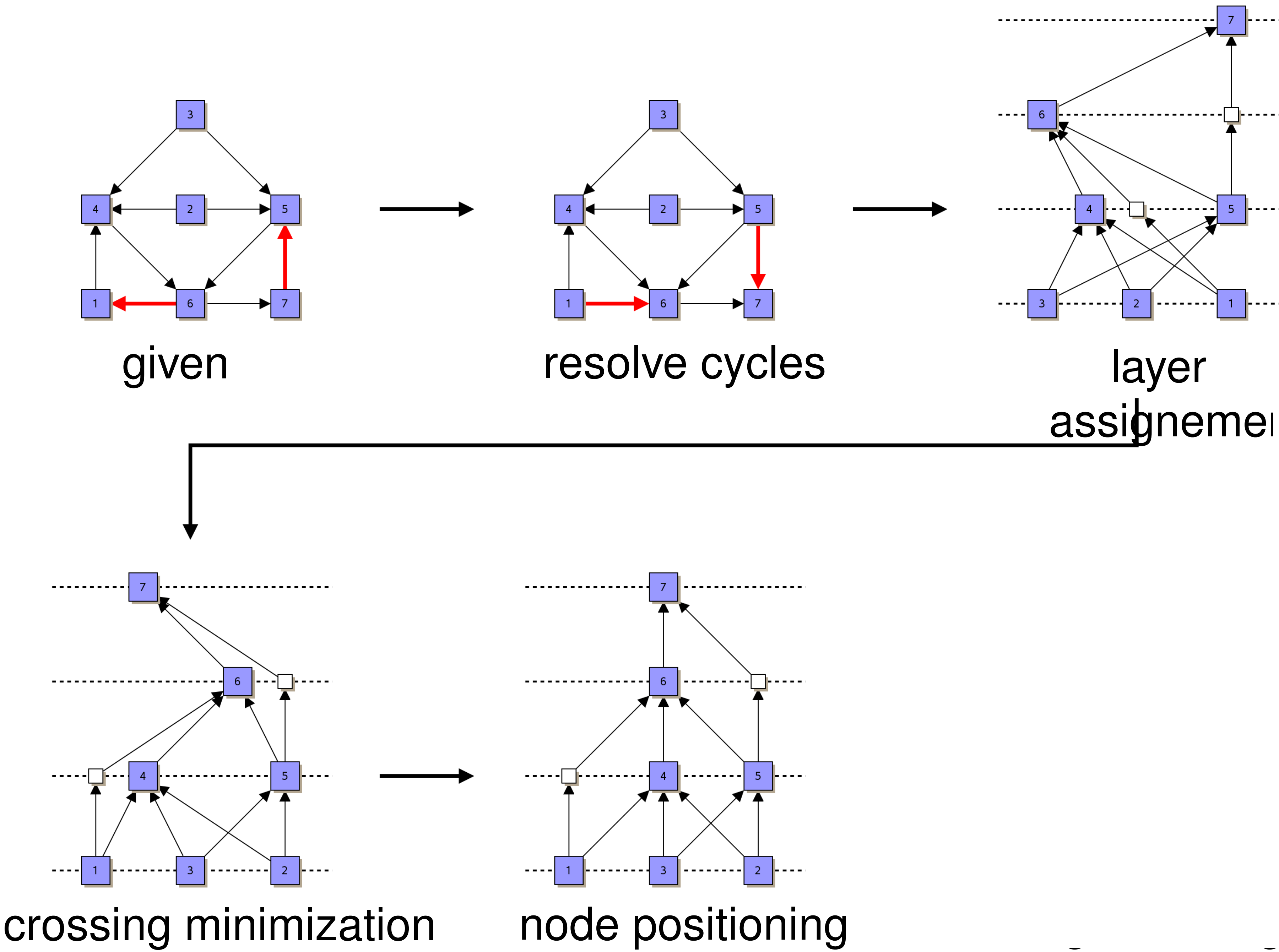


layer
assignemen

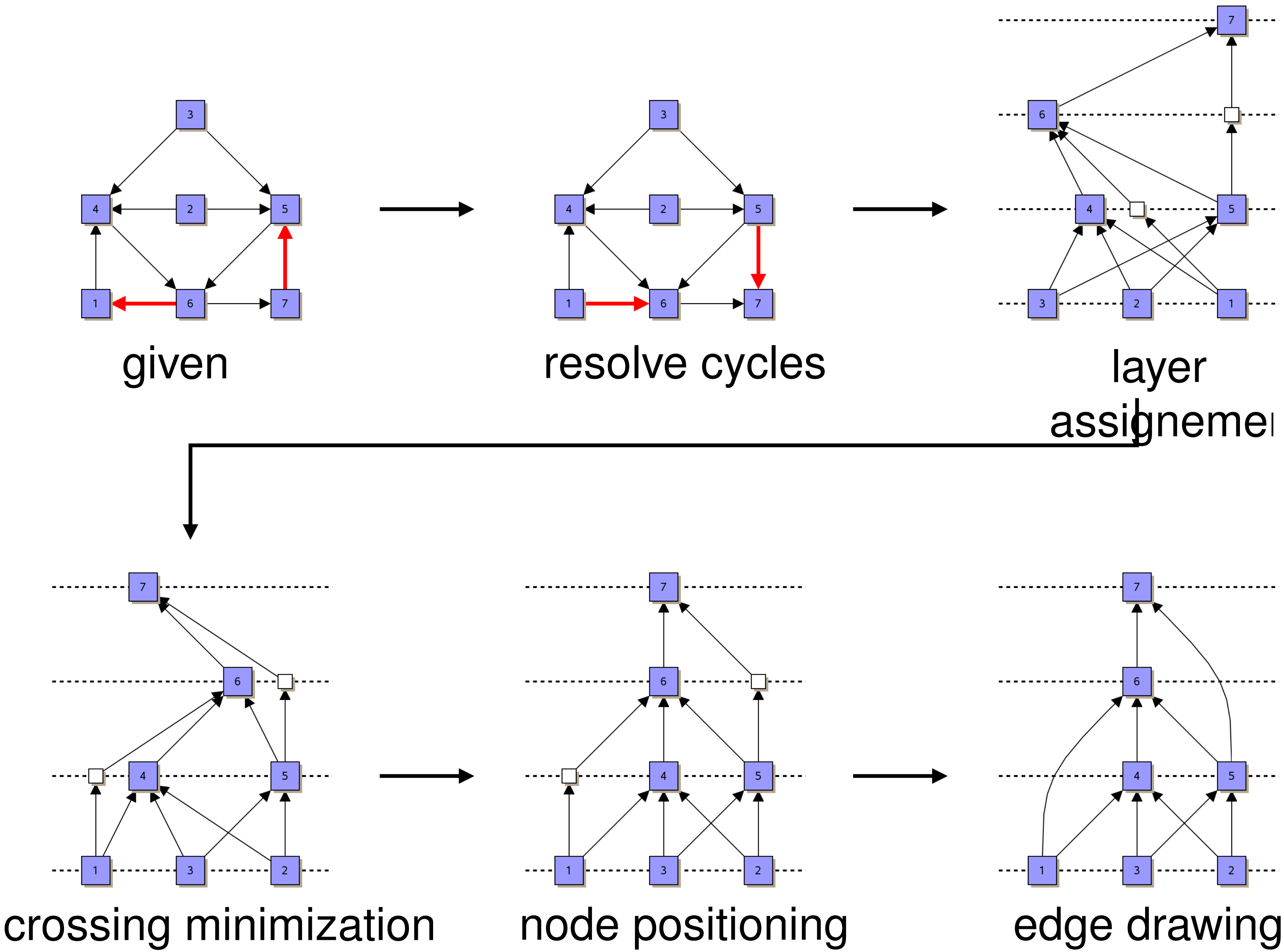


crossing minimizatio

Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)

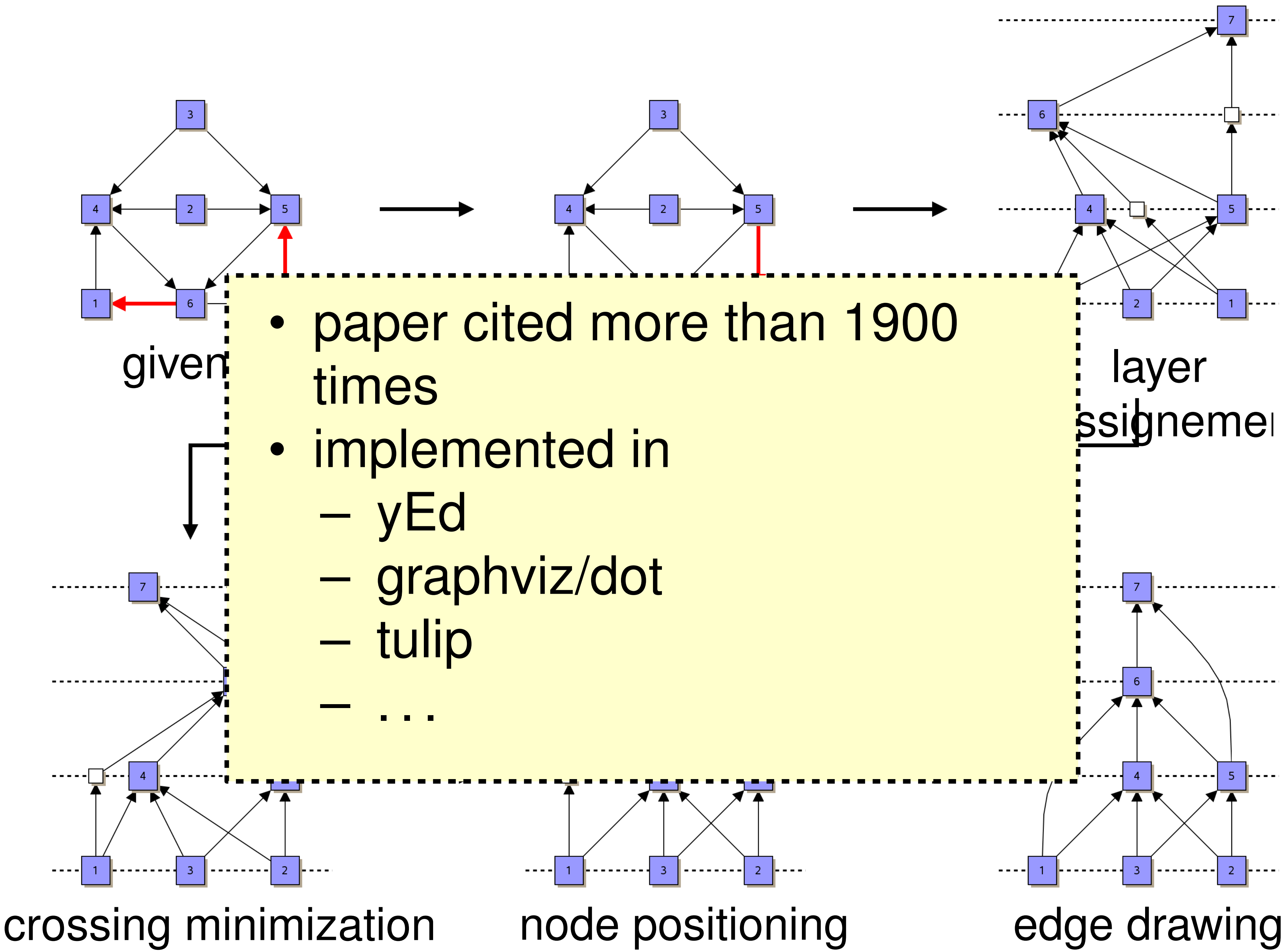


Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)

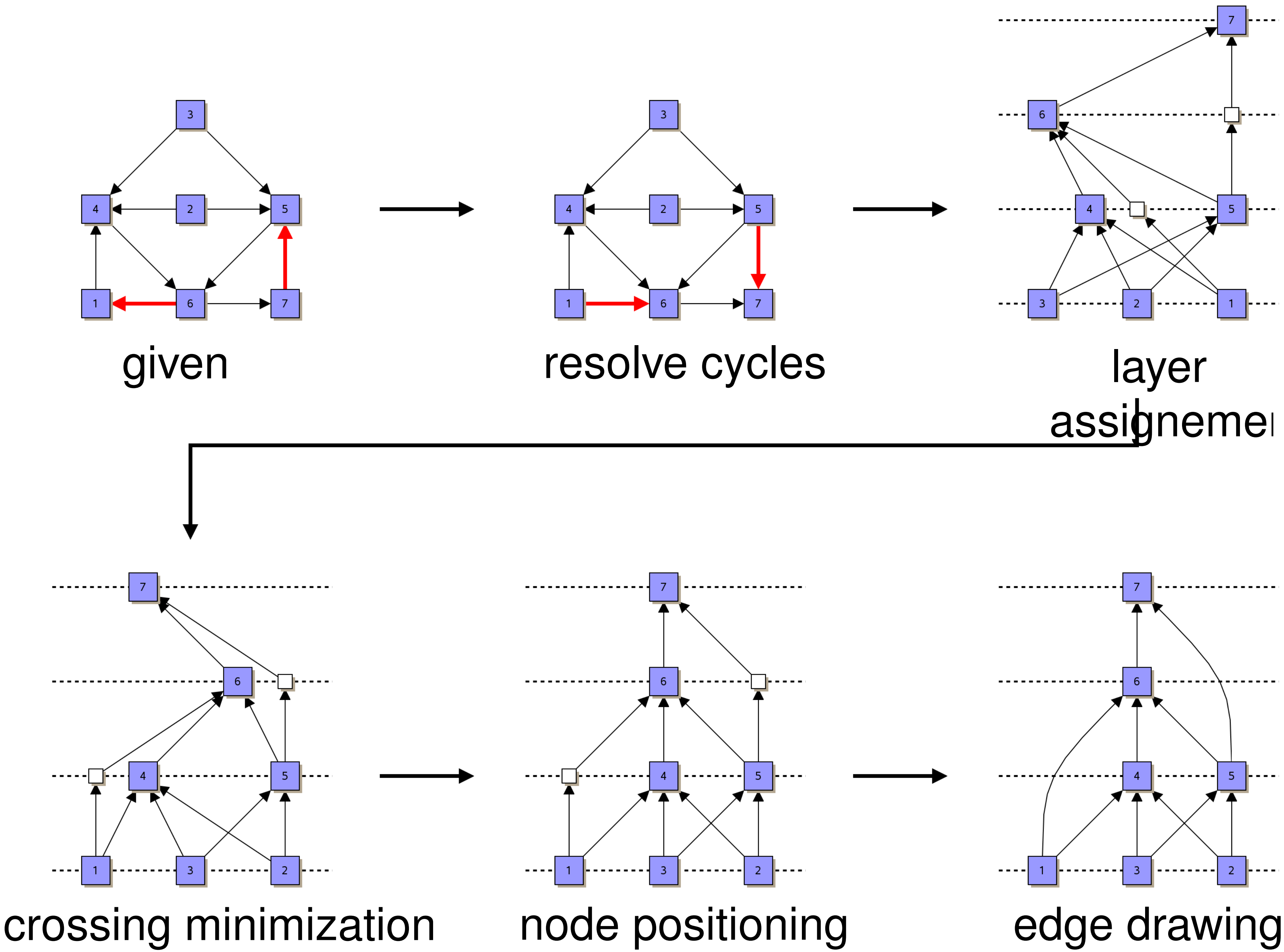


Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)

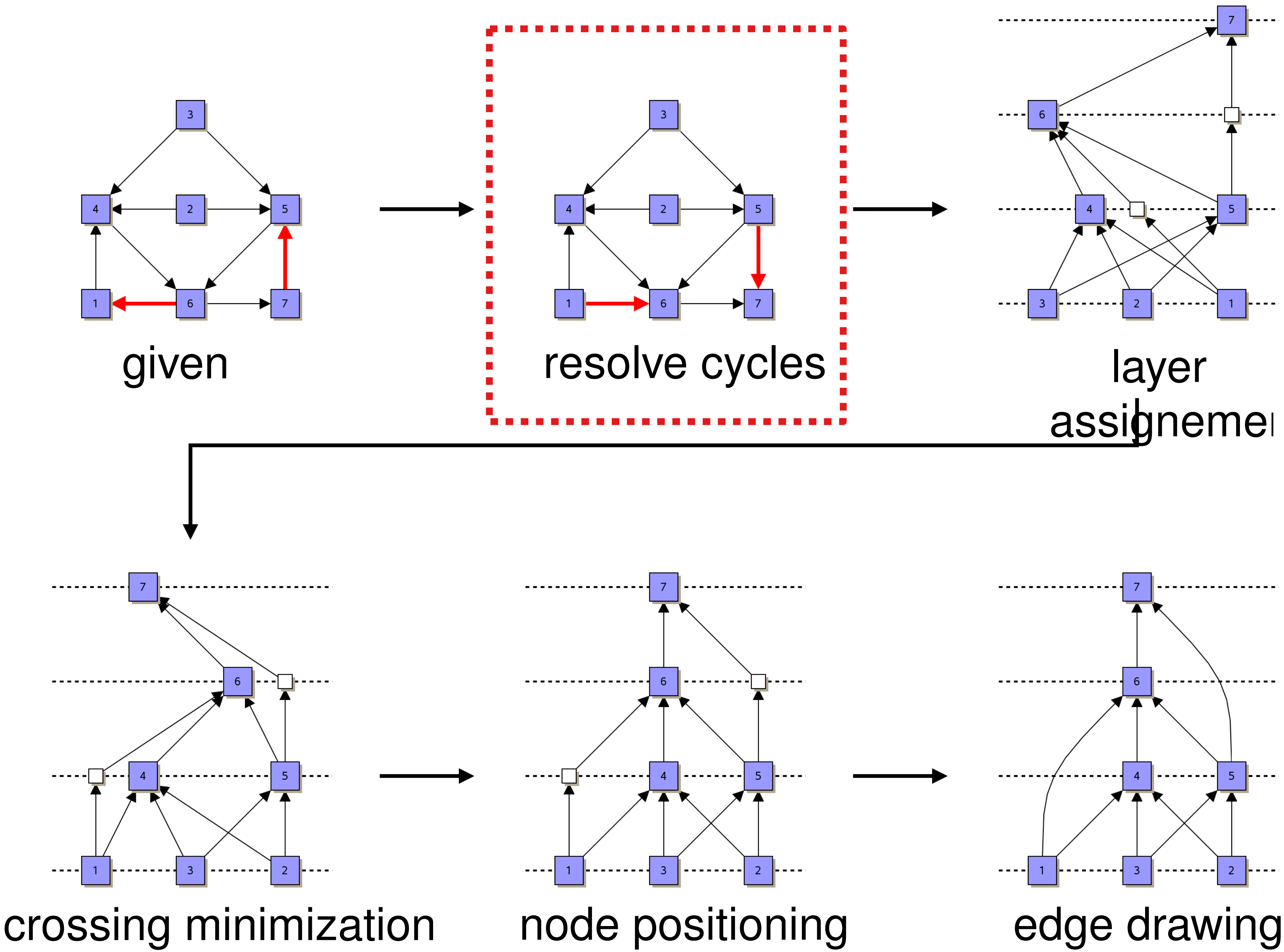
- paper cited more than 1900 times
- implemented in
  – yEd
  – graphviz/dot
  – tulip
  – . . .



Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Removing Cycles

Idea:
- find maximum acyclic subgraph
- inverse the directions of the other edges

Removing Cycles

Idea:
- find maximum acyclic subgraph
- inverse the directions of the other edges

MAXIMUM ACYCLIC SUBGRAPH

Given: directed graph $D = (V, A)$

Find: acyclic subgraph $D' = (V, A')$ with maximum $|A'|$

Removing Cycles

Idea:
- find maximum acyclic subgraph
- inverse the directions of the other edges

MAXIMUM ACYCLIC SUBGRAPH

Given: directed graph $D = (V, A)$

Find: acyclic subgraph $D' = (V, A')$ with maximum $|A'|$

MINIMUM FEEDBACK ARC SET (FAS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f)$ acyclic with minimum $|A_f|$

Removing Cycles

Idea:
- find maximum acyclic subgraph
- inverse the directions of the other edges

MAXIMUM ACYCLIC SUBGRAPH

Given: directed graph $D = (V, A)$

Find: acyclic subgraph $D' = (V, A')$ with maximum $|A'|$

MINIMUM FEEDBACK ARC SET (FAS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f)$ acyclic with minimum $|A_f|$

MINIMUM FEEDBACK SET (FS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \mathrm{rev}(A_f))$ acyclic with minimum $|A_f|$

Removing Cycles

Idea:
- find maximum acyclic subgraph
- inverse the directions of the other edges

MAXIMUM ACYCLIC SUBGRAPH

Given: directed graph $D = (V, A)$

Find: acyclic subgraph $D' = (V, A')$ with maximum $|A'|$

MINIMUM FEEDBACK ARC SET (FAS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f)$ acyclic with minimum $|A_f|$

MINIMUM FEEDBACK SET (FS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \text{rev}(A_f))$ acyclic with minimum $|A_f|$

Question: Is **FS** also a **FAS**? and the opposite?

Removing Cycles

Idea:
- find maximum acyclic subgraph
- inverse the directions of the other edges

MAXIMUM ACYCLIC SUBGRAPH

Given: directed graph $D = (V, A)$

Find: acyclic subgraph $D' = (V, A')$ with maximum $|A'|$

MINIMUM FEEDBACK ARC SET (FAS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f)$ acyclic with minimum $|A_f|$

MINIMUM FEEDBACK SET (FS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \text{rev}(A_f))$ acyclic with minimum $|A_f|$

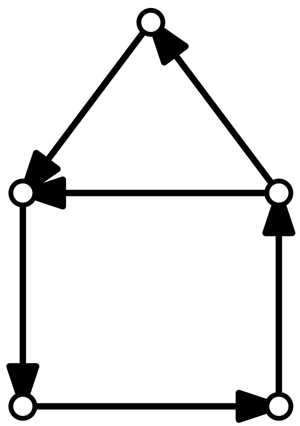
All three problems are NP-hard!

Trivial Heuristic

MINIMUM FEEDBACK SET (FS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \mathrm{rev}(A_f))$ acyclic with minimum $|A_f|$

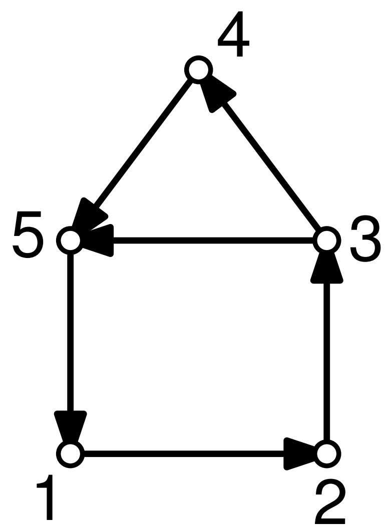


Trivial Heuristic

MINIMUM FEEDBACK SET (FS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \mathrm{rev}(A_f))$ acyclic with minimum $|A_f|$

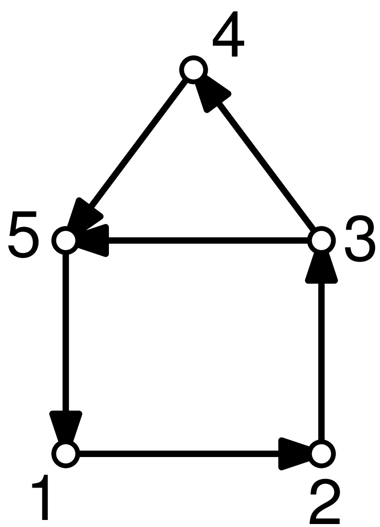


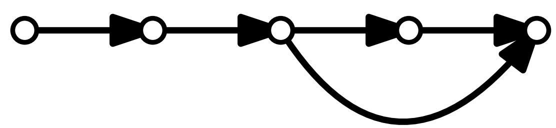
Trivial Heuristic

Minimum Feedback Set (FS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \mathrm{rev}(A_f))$ acyclic with minimum $|A_f|$





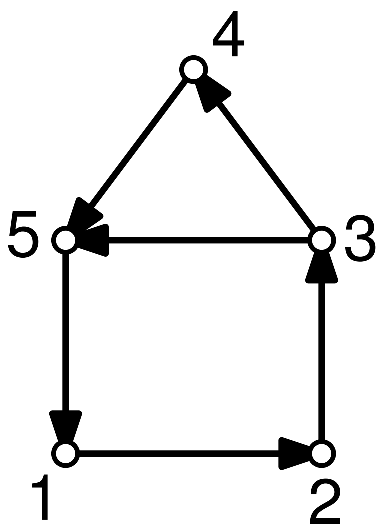
place vertices in order

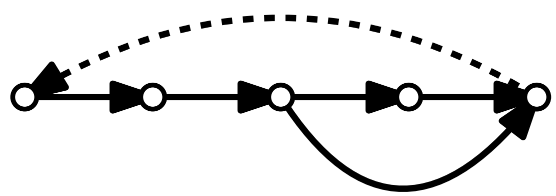
Trivial Heuristic

MINIMUM FEEDBACK SET (FS)

Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \mathrm{rev}(A_f))$ acyclic with minimum $|A_f|$





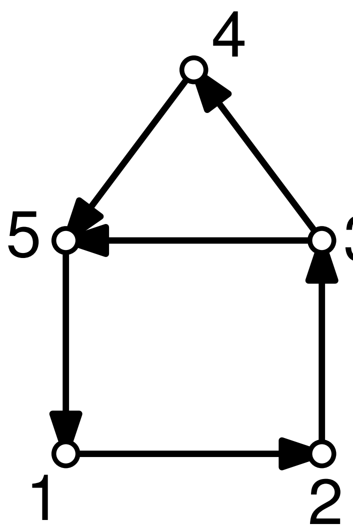
place vertices in order

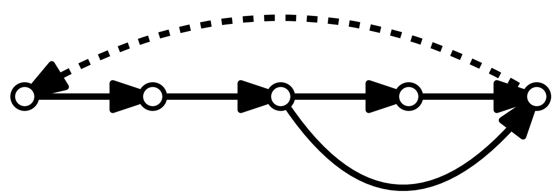
Trivial Heuristic

MINIMUM FEEDBACK SET (FS)

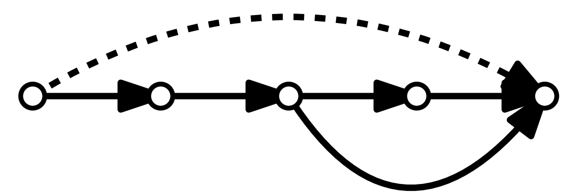
Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \mathrm{rev}(A_f))$ acyclic with minimum $|A_f|$





place vertices in order



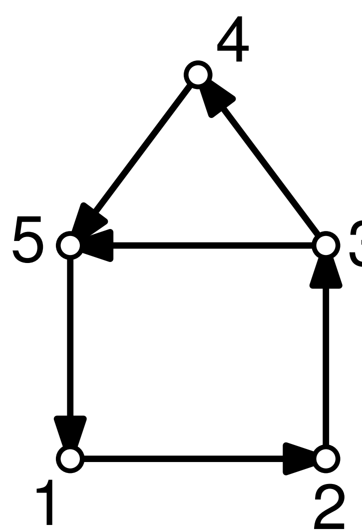
reverse all the backward edges

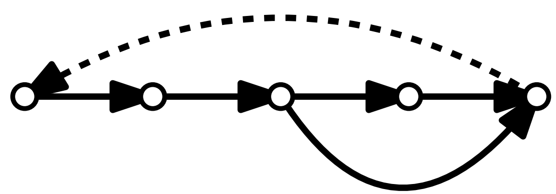
Trivial Heuristic

MINIMUM FEEDBACK SET (FS)

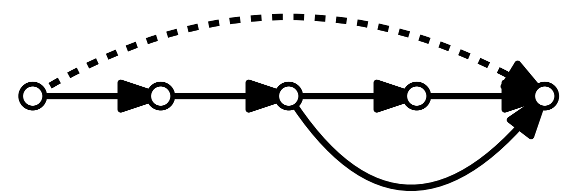
Given: directed graph $D = (V, A)$

Find: $A_f \subset A$, with $D_f = (V, A \setminus A_f \cup \mathrm{rev}(A_f))$ acyclic with minimum $|A_f|$





place vertices in order



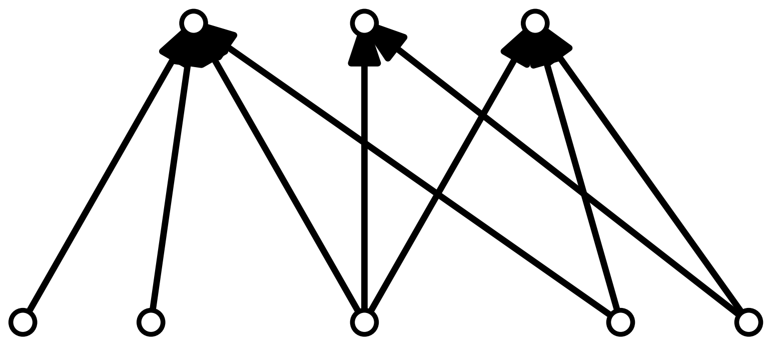
reverse all the backward edges

Benefit: Very simple to implement

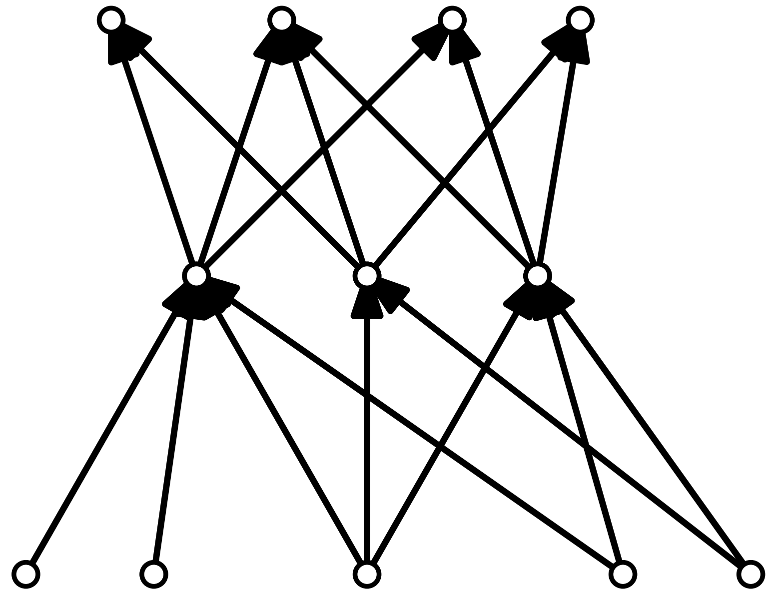
Drawback: No guarantee on the number of reversed edges

Heuristic with guarantees (Eades, Lin, Smyth 1993)

source – no incomming edge
sink – no outgoing edge



no cycles if all vertices are only sources and sinks



no cycles if graph can be eliminated by always removing either sources or sinks

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

Heuristic with guarantees (Eades, Lin, Smyth 1993)

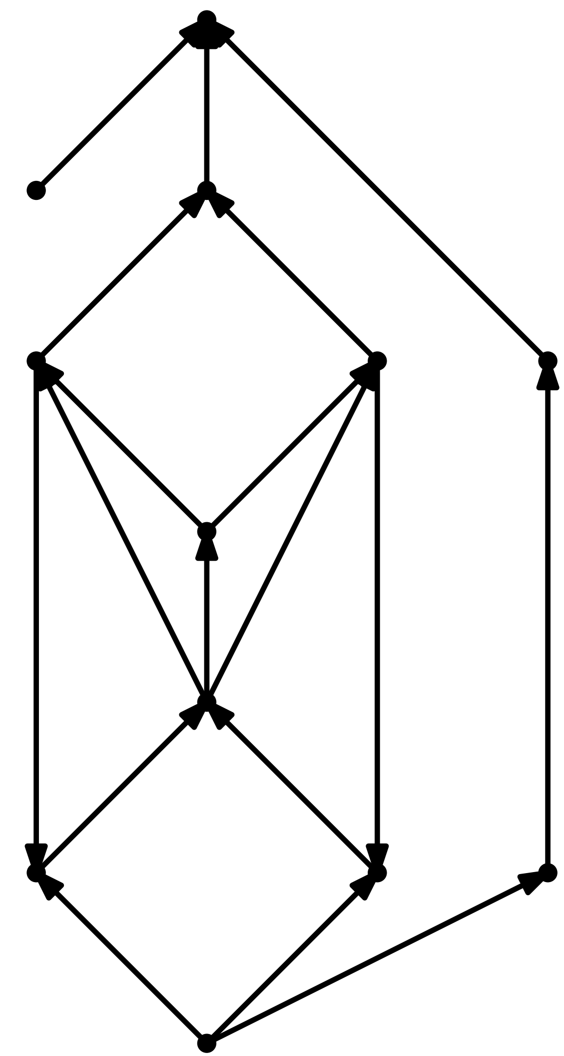
$A' := \emptyset$;

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

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

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

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



Heuristic with guarantees (Eades, Lin, Smyth 1993)

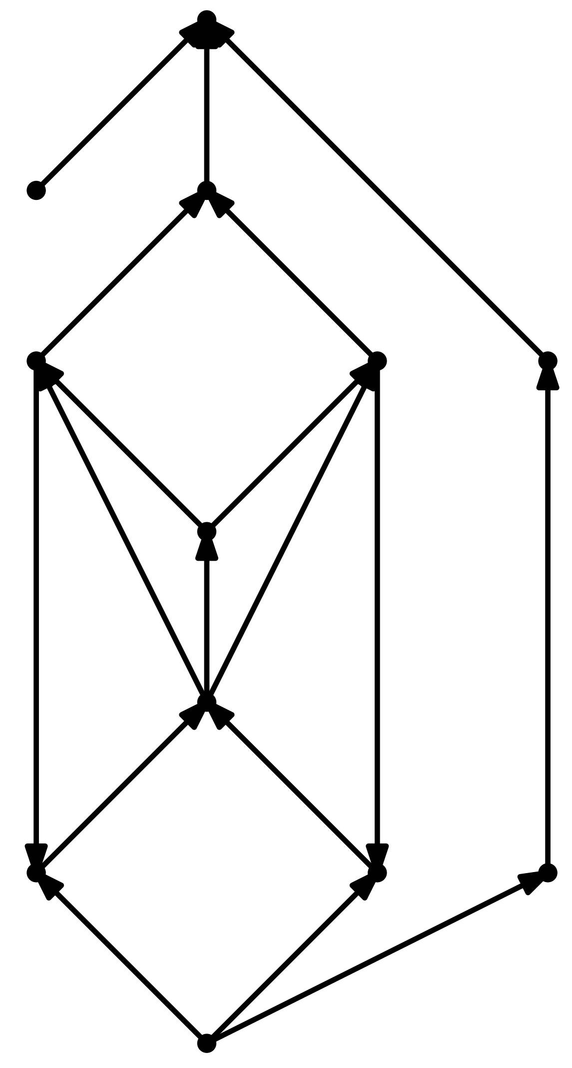
$A' := \emptyset$;

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

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

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

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



Heuristic with guarantees (Eades, Lin, Smyth 1993)

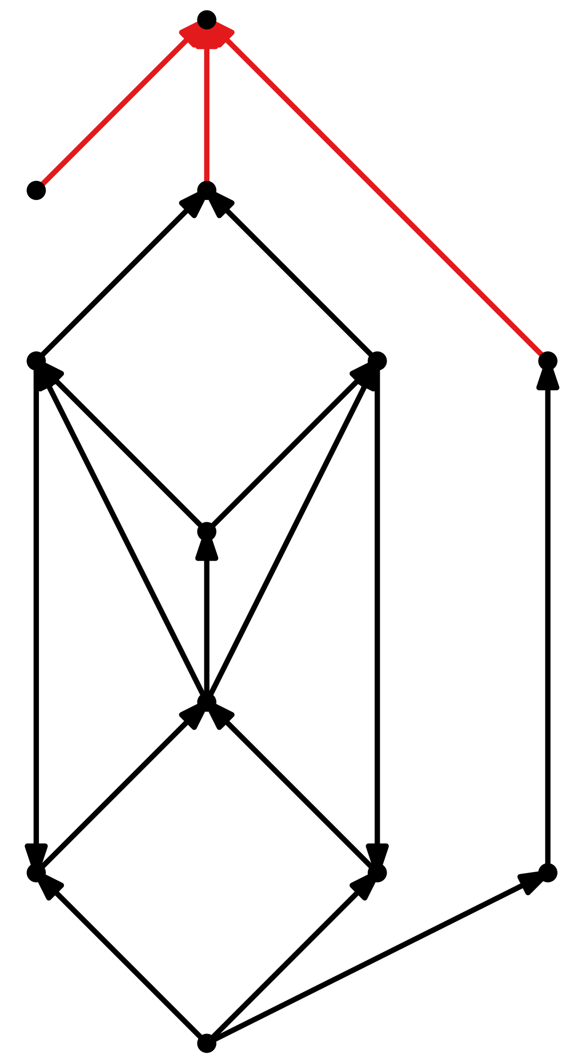
$A' := \emptyset$;

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

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

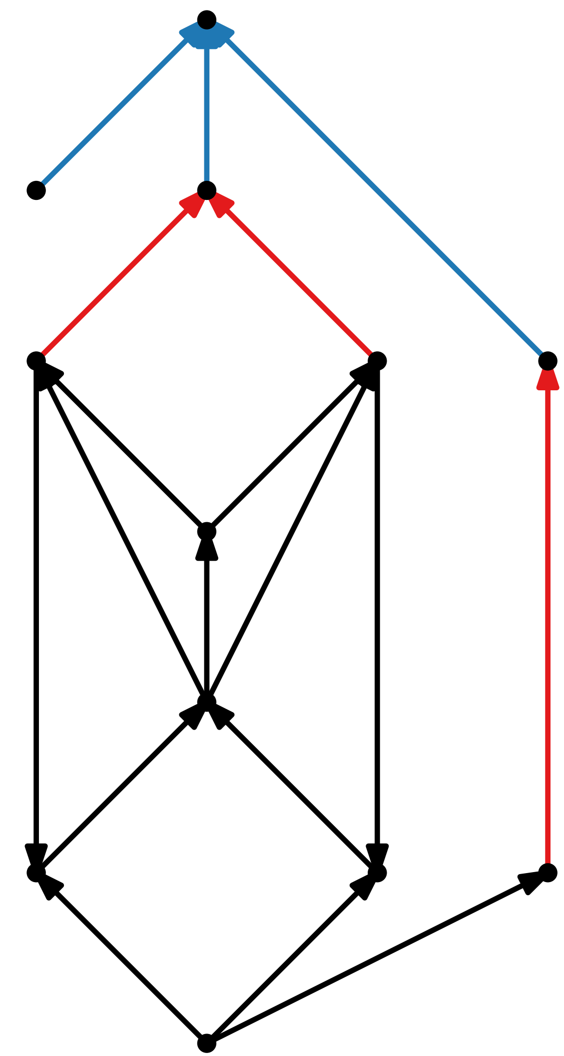
    $A' \leftarrow A' \cup N^{\leftarrow}(v)$

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



Heuristic with guarantees (Eades, Lin, Smyth 1993)

```
A' := ∅;
while V ≠ ∅ do
    while in V exists a sink v do
        A' ← A' ∪ N←(v)
        remove v and N←(v)
```



Heuristic with guarantees (Eades, Lin, Smyth 1993)

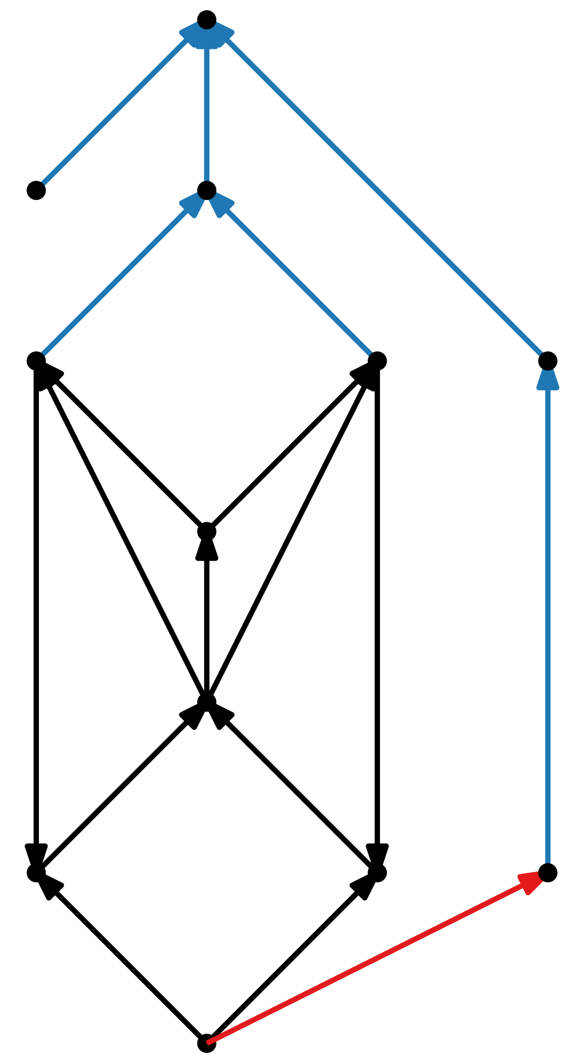
$A' := \emptyset$;

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

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

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

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



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

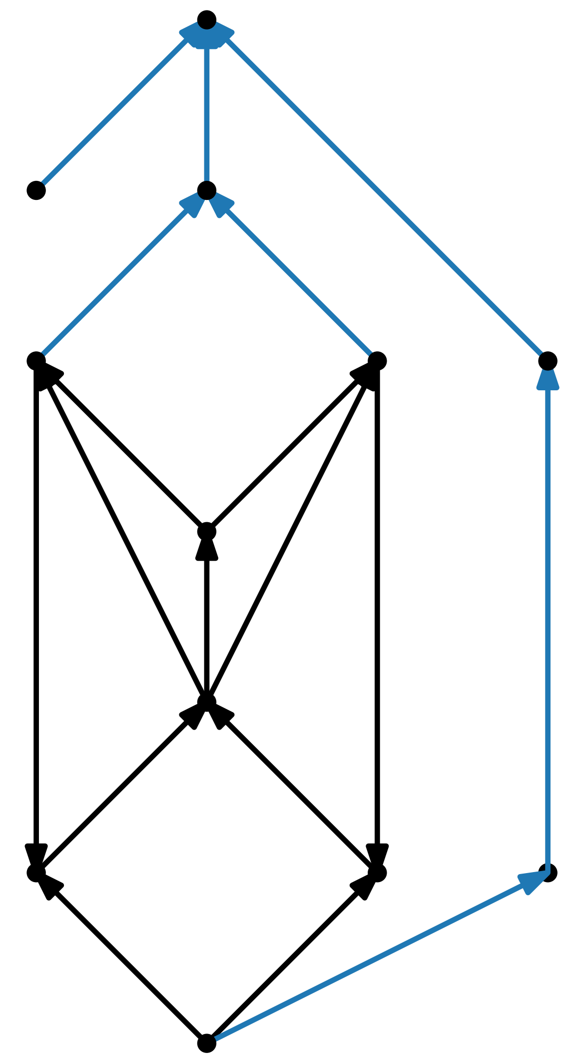
while $V \neq \emptyset$ **do**

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

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

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

  Remove all isolated node from V



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

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

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

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

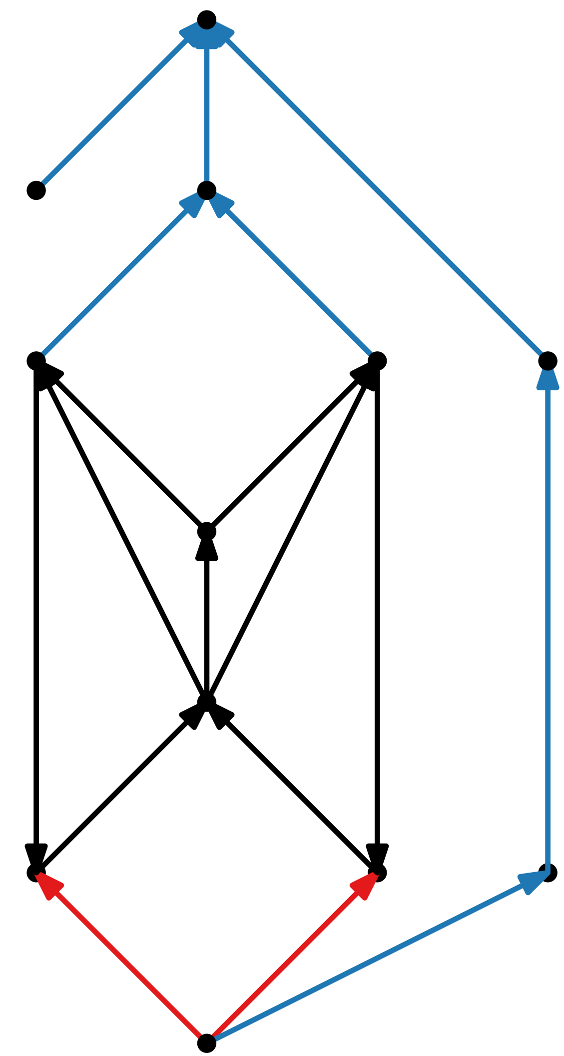
    remove v and $N^{\leftarrow}(v)$

  Remove all isolated node from V

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

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

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



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

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

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

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

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

  Remove all isolated node from V

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

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

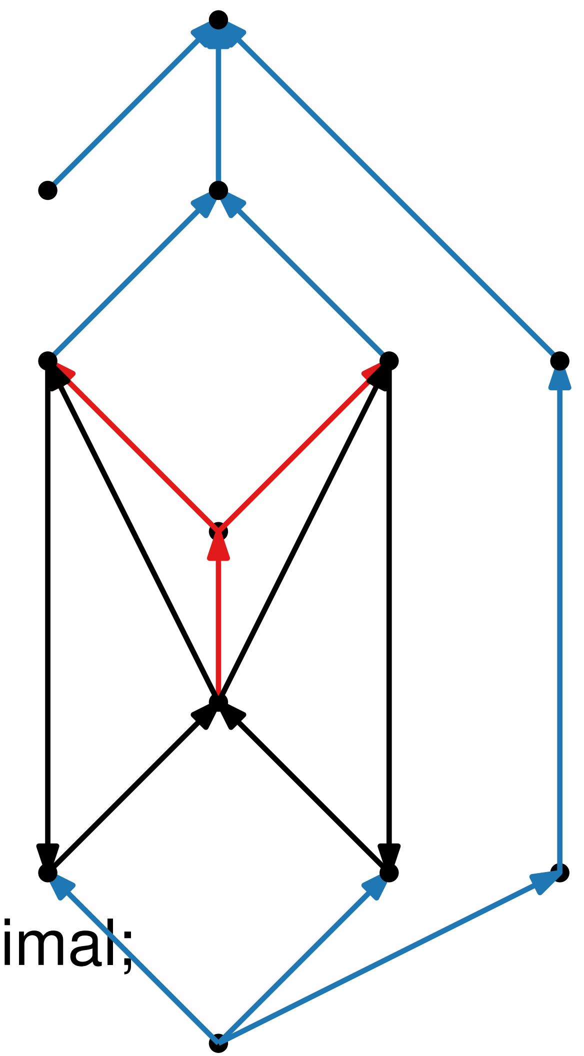
    remove v and $N^{\rightarrow}(v)$

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

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

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

    remove v and $N(v)$



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

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

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

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

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

    Remove all isolated node from V

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

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

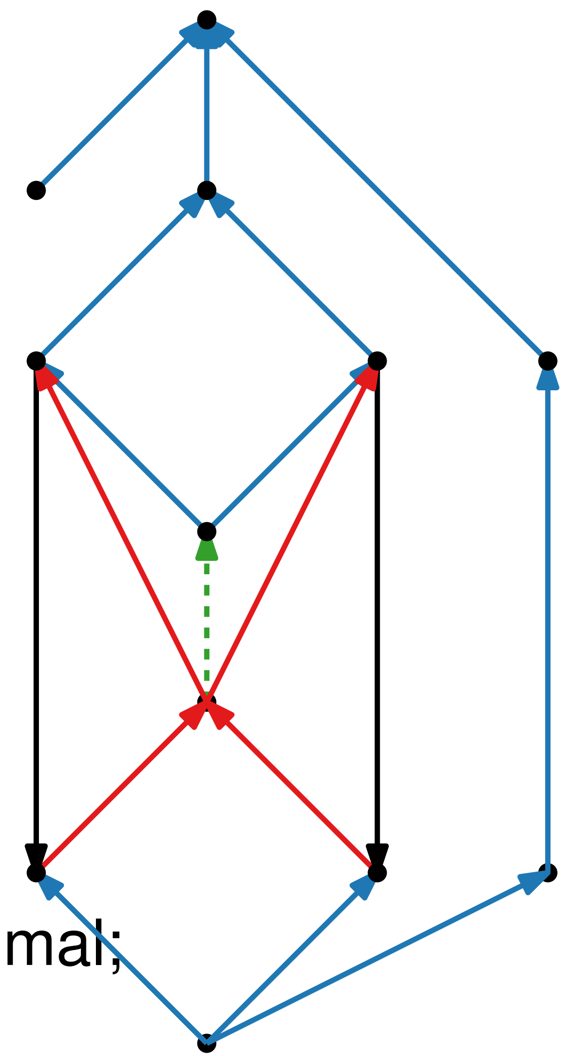
        remove v and $N^{\rightarrow}(v)$

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

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

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

        remove v and $N(v)$



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

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

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

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

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

  Remove all isolated node from V

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

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

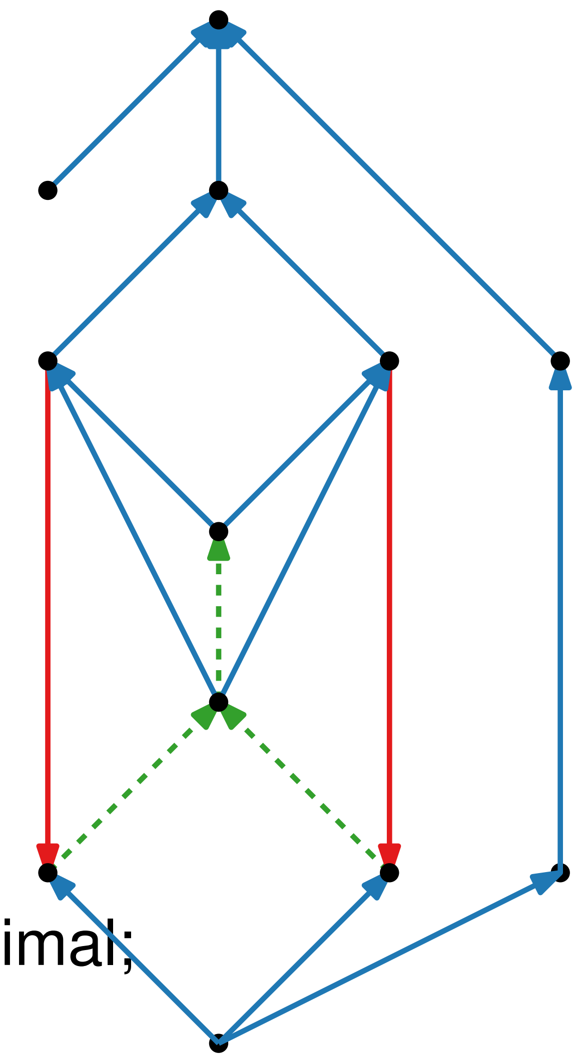
    remove v and $N^{\rightarrow}(v)$

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

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

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

    remove v and $N(v)$



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

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

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

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

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

  Remove all isolated node from V

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

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

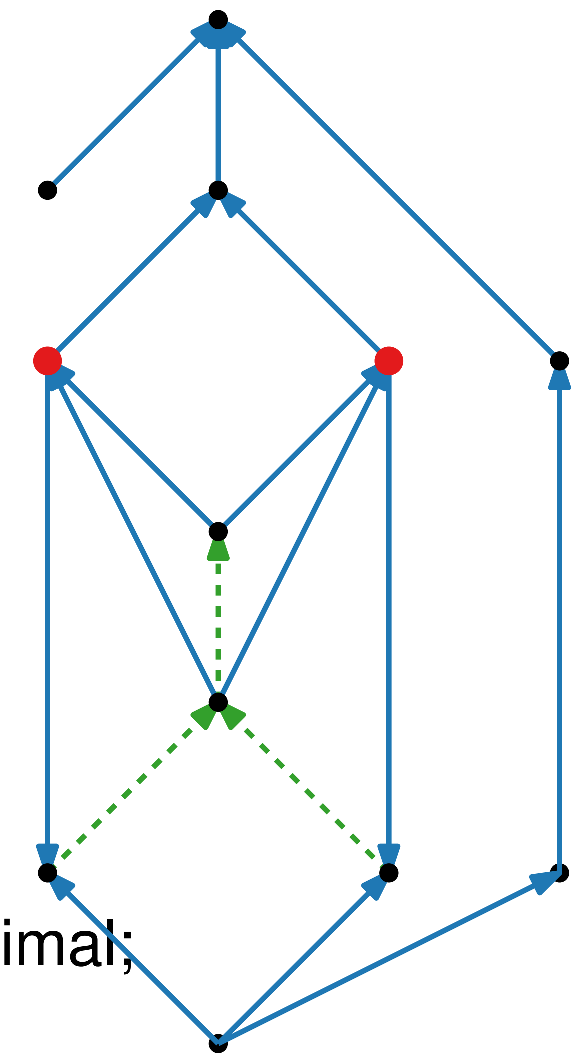
    remove v and $N^{\rightarrow}(v)$

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

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

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

    remove v and $N(v)$



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

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

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

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

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

  Remove all isolated node from V

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

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

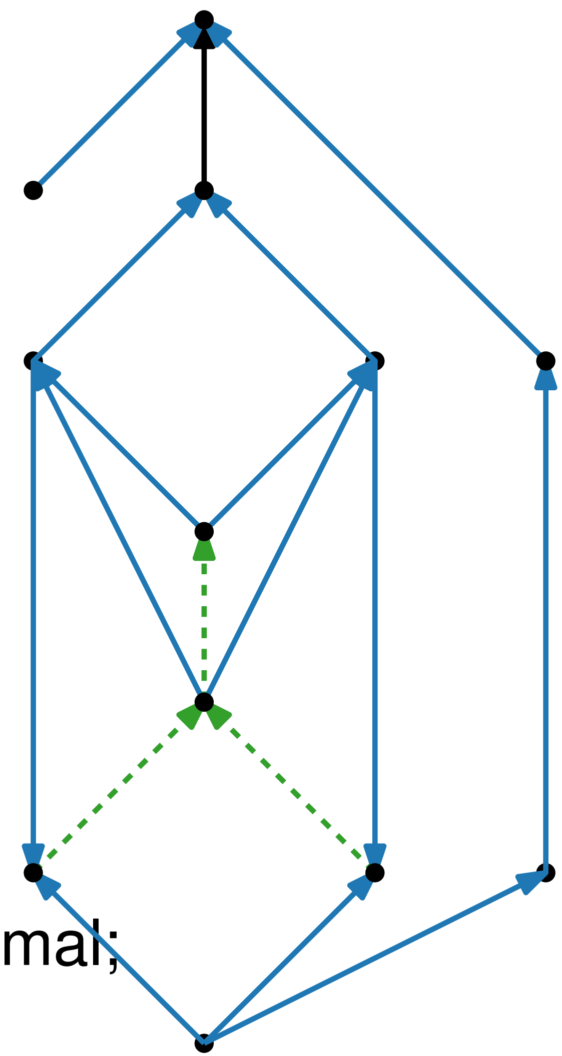
    remove v and $N^{\rightarrow}(v)$

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

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

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

    remove v and $N(v)$



Heuristic with guarantees (Eades, Lin, Smyth 1993)

$A' := \emptyset$;

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

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

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

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

    Remove all isolated node from V

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

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

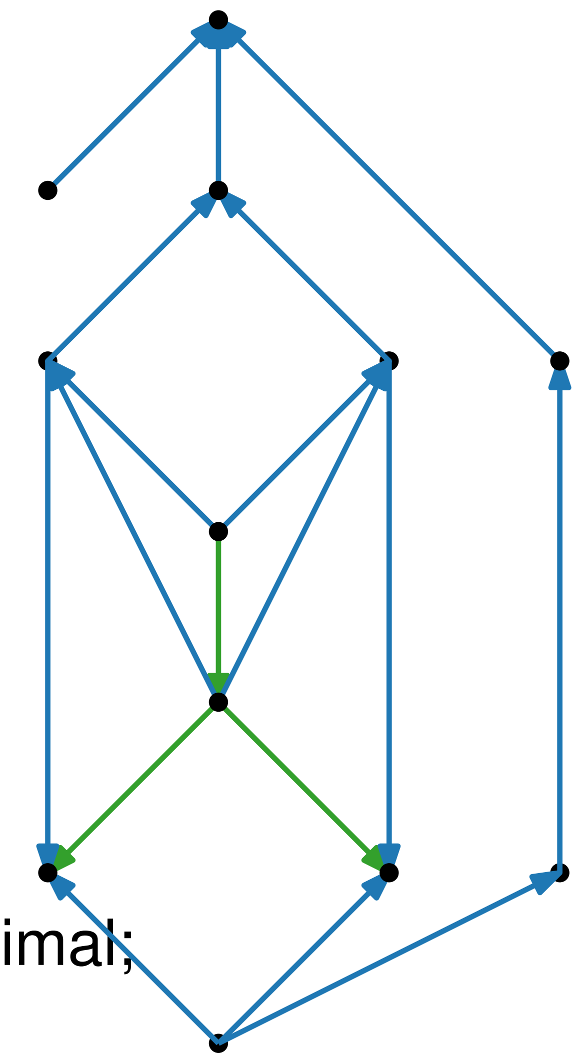
        remove v and $N^{\rightarrow}(v)$

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

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

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

        remove v and $N(v)$



Guarantees and related work

Theorem: Let $D = (V, A)$ be a connected, directed graph without 2-cycles. Heuristic of Eades at al. computes a set of edges A' with $|A'| \geq |A|/2 + |V|/6$.
The running time is $O(|A|)$.

Guarantees and related work

Theorem: Let $D = (V, A)$ be a connected, directed graph without 2-cycles. Heuristic of Eades at al. computes a set of edges A' with $|A'| \geq |A|/2 + |V|/6$. The running time is $O(|A|)$.

Many other heuristics:

- based on vertex ordering – use DFS (Rowe et al. 1987)

Guarantees and related work

Theorem: Let $D = (V, A)$ be a connected, directed graph without 2-cycles. Heuristic of Eades at al. computes a set of edges A' with $|A'| \geq |A|/2 + |V|/6$. The running time is $O(|A|)$.

Many other heuristics:

- based on vertex ordering – use DFS (Rowe et al. 1987)
- based on cycle braking – start with an empty edge set and only add edges as long as they do not create cycles

Guarantees and related work

Theorem: Let $D = (V, A)$ be a connected, directed graph without 2-cycles. Heuristic of Eades at al. computes a set of edges A' with $|A'| \geq |A|/2 + |V|/6$.
The running time is $O(|A|)$.

Many other heuristics:

- based on vertex ordering – use DFS (Rowe et al. 1987)
- based on cycle braking – start with an empty edge set and only add edges as long as they do not create cycles
- or remove edges that participate in many cycles, again with a use of DFS (Gansner et al. 1993) – performs well and is implemented in Graphvis

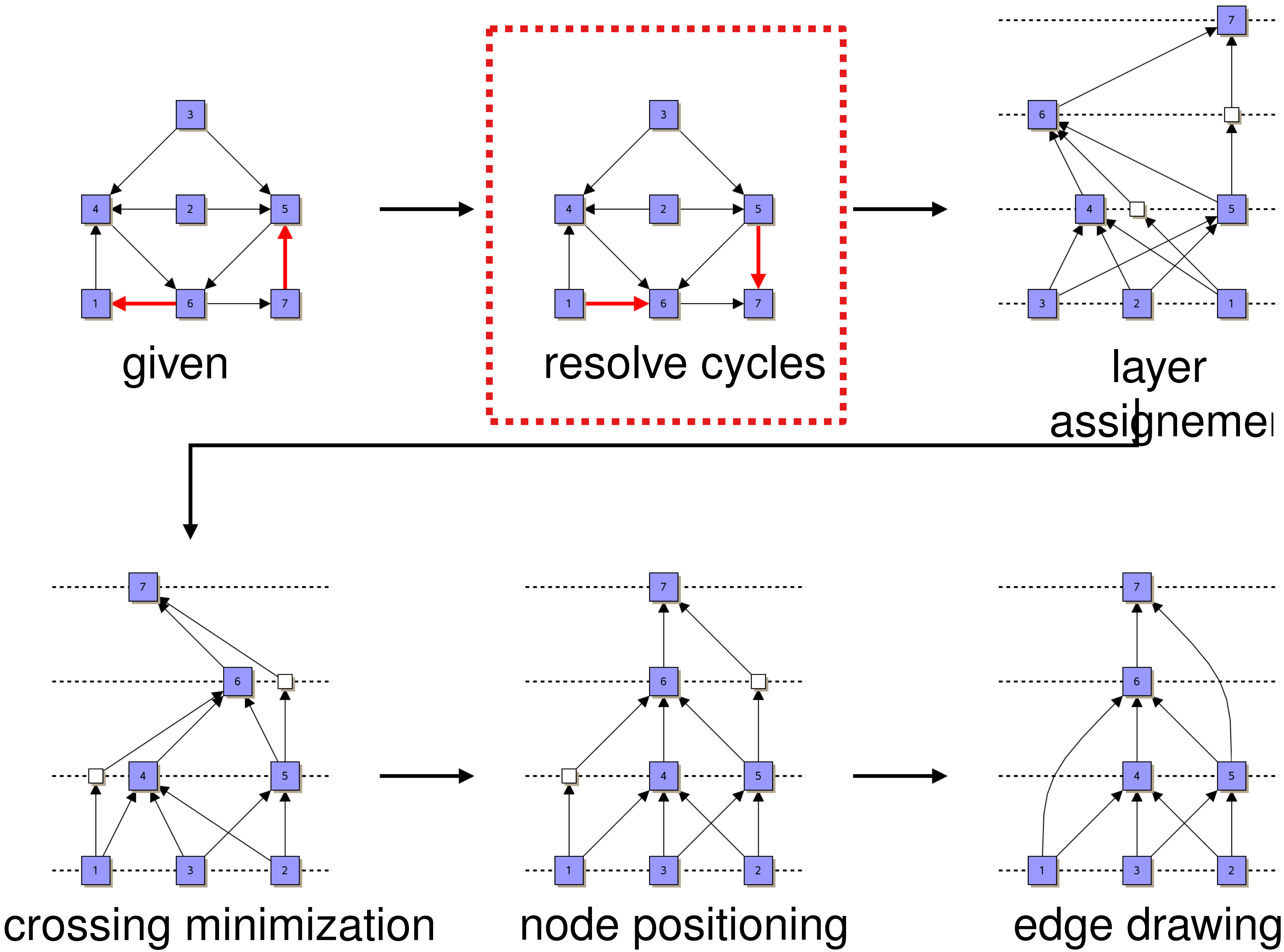
Guarantees and related work

Theorem: Let $D = (V, A)$ be a connected, directed graph without 2-cycles. Heuristic of Eades at al. computes a set of edges A' with $|A'| \geq |A|/2 + |V|/6$. The running time is $O(|A|)$.

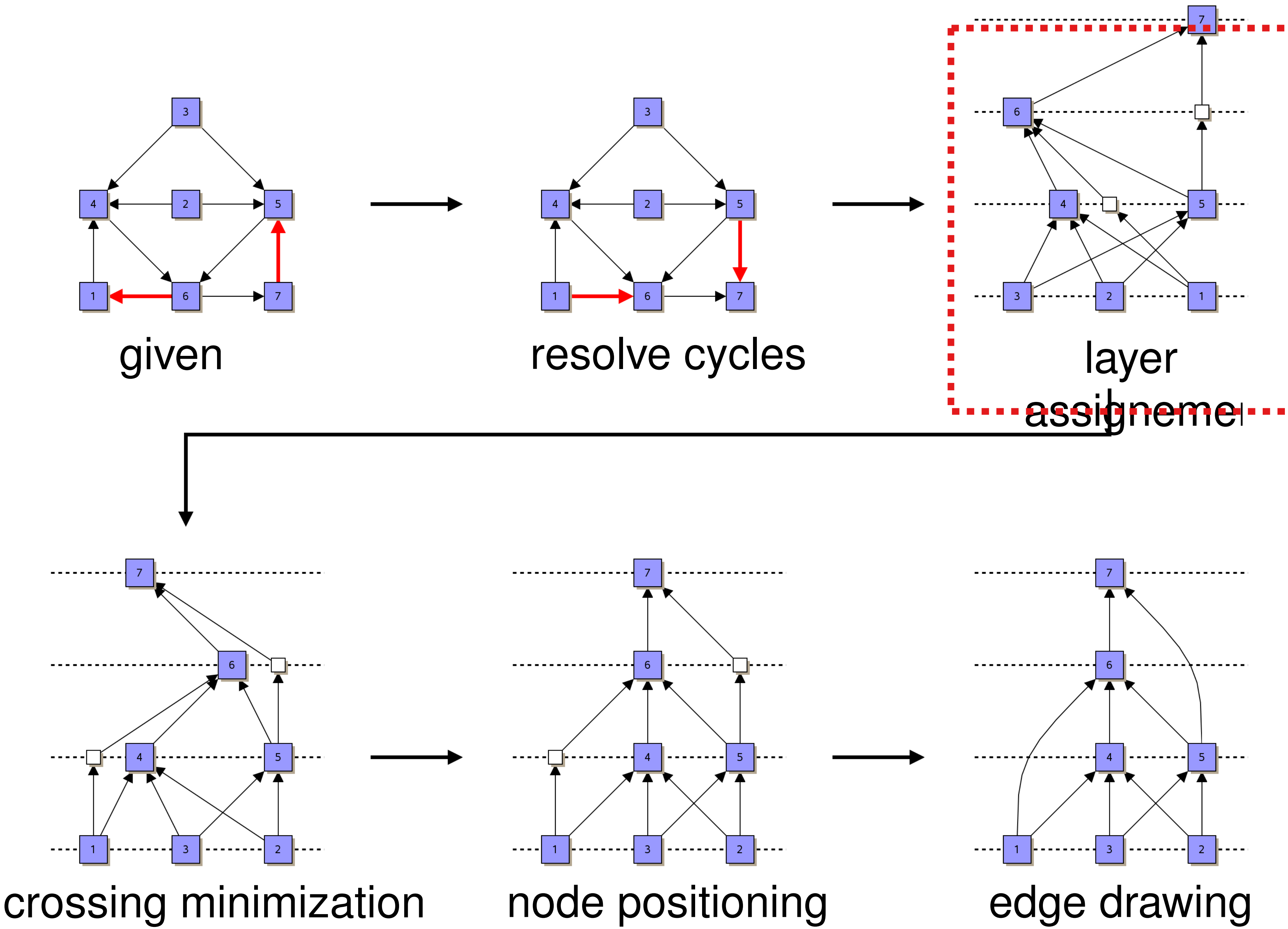
Many other heuristics:

- based on vertex ordering – use DFS (Rowe et al. 1987)
- based on cycle braking – start with an empty edge set and only add edges as long as they do not create cycles
- or remove edges that participate in many cycles, again with a use of DFS (Gansner et al. 1993) – performs well and is implemented in Graphvis
- optimal solution integer linear programming, using branch-and-cut technique (Grötschel et al. 1985)

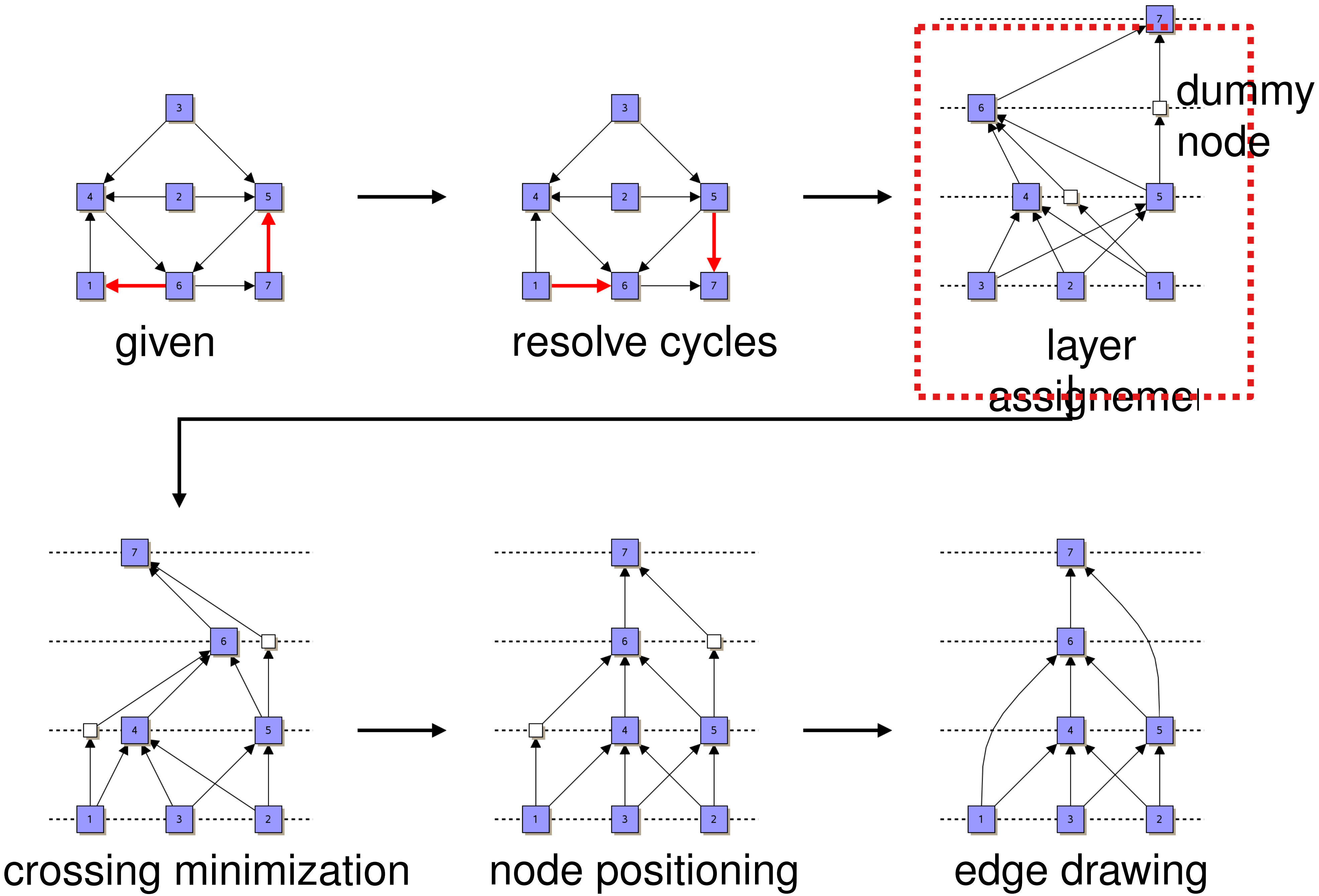
Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Layer Assignement

Given.: directed acyclic graph (DAG) $D = (V, A)$

Find: Partition the vertex set V into disjoint subsets (**layers**) $L_1, \ldots, L_h$ s.t. $(u, v) \in A,\ u \in L_i,\ v \in L_j \Rightarrow i < j$

Def: y-Coordinate $y(u) = i \Leftrightarrow u \in L_i$

Layer Assignement

Given.: directed acyclic graph (DAG) $D = (V, A)$

Find: Partition the vertex set V into disjoint subsets (**layers**) $L_1, \ldots, L_h$ s.t. $(u, v) \in A,\ u \in L_i,\ v \in L_j \Rightarrow i < j$

Def: y-Coordinate $y(u) = i \Leftrightarrow u \in L_i$

Think for a minute and then share

What could we optimize when doing the layer assignment?

Layer Assignement

Given.: directed acyclic graph (DAG) $D = (V, A)$

Find: Partition the vertex set V into disjoint subsets (**layers**) $L_1, \dots, L_h$ s.t. $(u, v) \in A$, $u \in L_i$, $v \in L_j \Rightarrow i < j$

Def: y-Coordinate $y(u) = i \Leftrightarrow u \in L_i$

Possible optimization criteria

- minimize the number of layers h (= height of the layouts)
- minimize the total length of edges ($\approx$ number of dummy nodes)
- minimize width, e.g. $\max\{|L_i| \mid 1 \leq i \leq h\}$

Layer Assignement

Given.: directed acyclic graph (DAG) $D = (V, A)$

Find: Partition the vertex set V into disjoint subsets (**layers**) $L_1, \dots, L_h$ s.t. $(u, v) \in A$, $u \in L_i$, $v \in L_j \Rightarrow i < j$

Def: y-Coordinate $y(u) = i \Leftrightarrow u \in L_i$

Possible optimization criteria

- minimize the number of layers h (= height of the layouts)
- minimize the total length of edges ($\approx$ number of dummy nodes)
- minimize width, e.g. $\max\{|L_i| \mid 1 \leq i \leq h\}$

Height Optimization

Idea: assign each node *v* to the layer L_i, where *i* is the length of the longest simple path from a source to *v*

- all incomming neighbours lie below *v*
- the resulting height *h* is minimized

Height Optimization

Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \dots, L_h$
$S :=$ sources of G
$i = 1;$

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i++$
  $G := G \setminus S$
  $S :=$ sources of G

Height Optimization

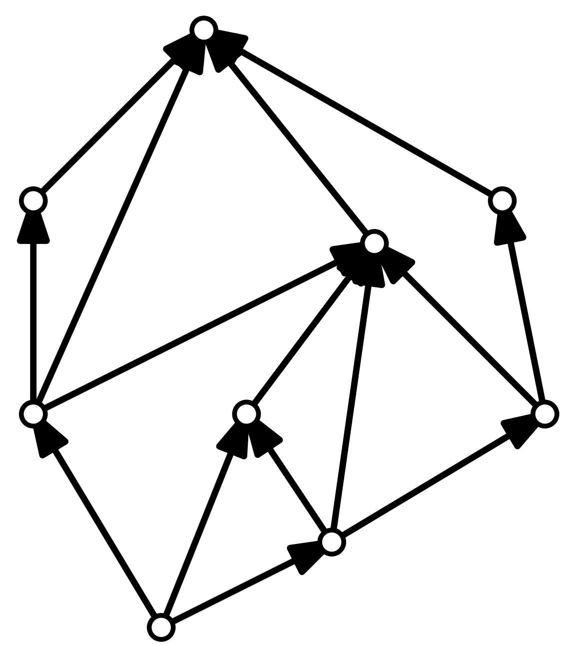
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \ldots, L_h$
S := sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i{+}{+}$
  $G := G \setminus S$
  S := sources of G



Height Optimization

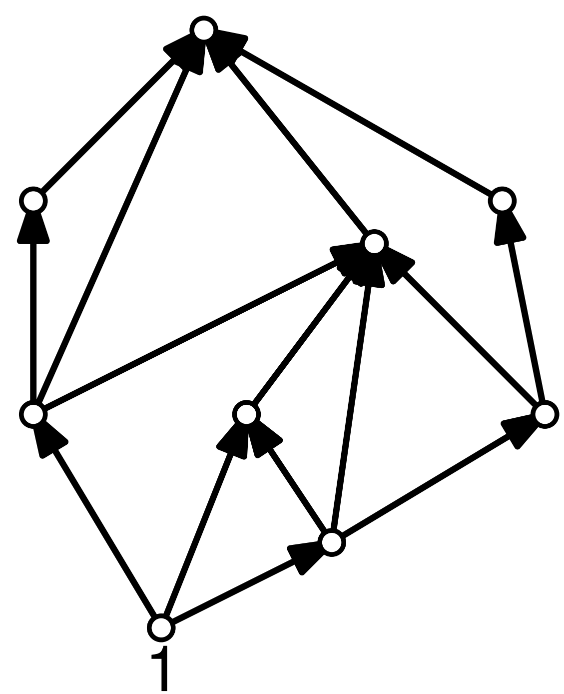
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \ldots, L_h$
S := sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i++$
  $G := G \setminus S$
  S := sources of G



Height Optimization

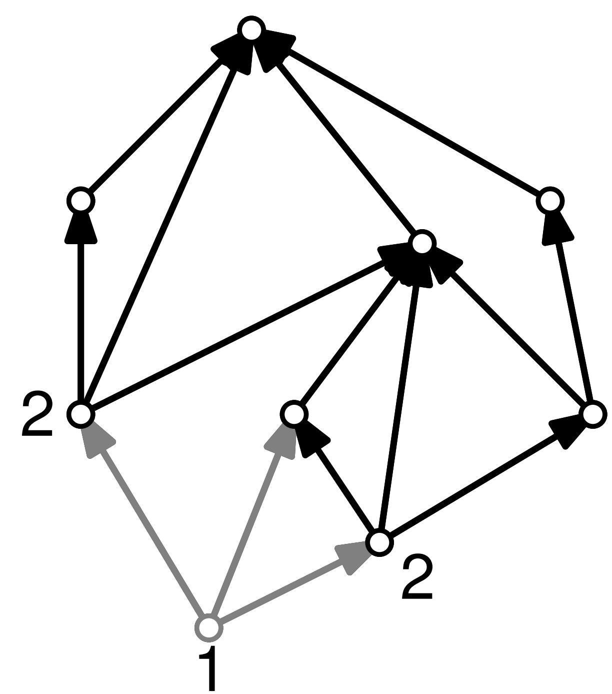
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \dots, L_h$
S := sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i{+}{+}$
  $G := G \setminus S$
  S := sources of G



Height Optimization

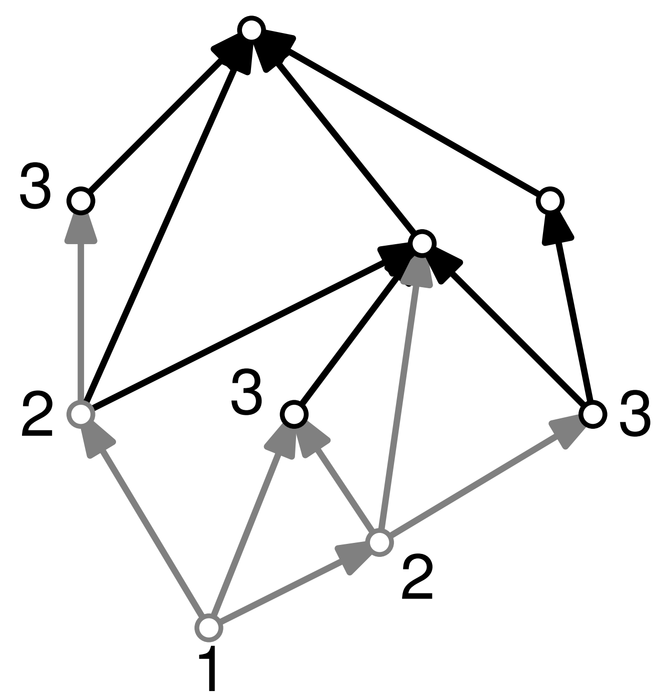
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \dots, L_h$
$S :=$ sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i++$
  $G := G \setminus S$
  $S :=$ sources of G



Height Optimization

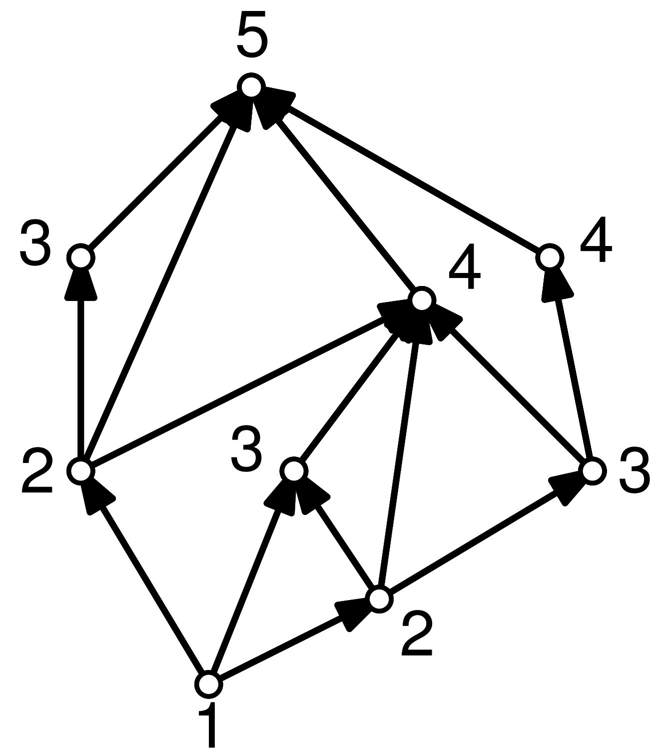
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \ldots, L_h$
$S :=$ sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i++$
  $G := G \setminus S$
  $S :=$ sources of G



Height Optimization

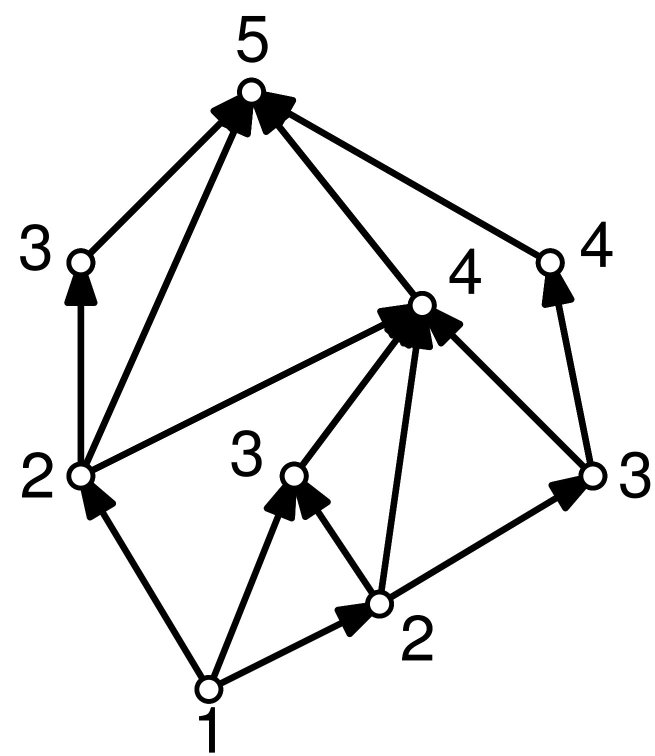
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \dots, L_h$
S := sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i{+}{+}$
  $G := G \setminus S$
  S := sources of G



Height Optimization

Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

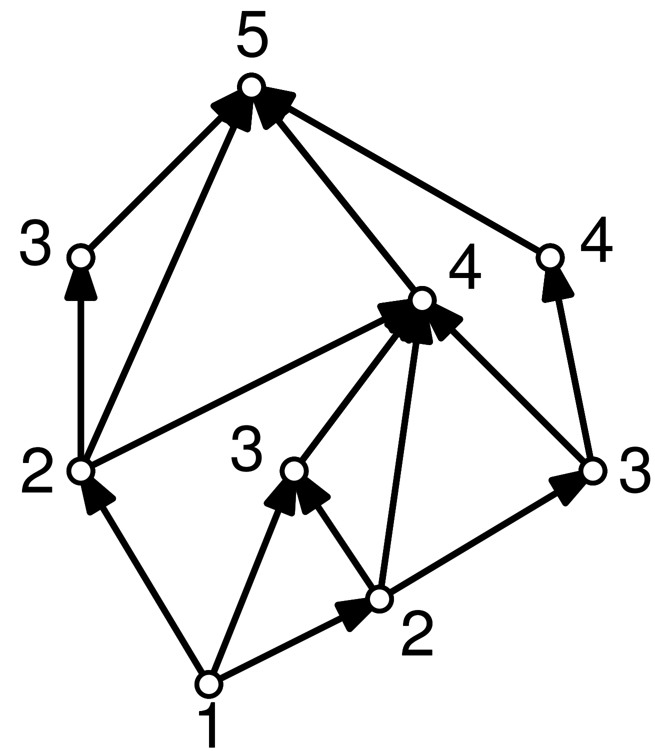
- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

also known as *topological numbering*

Input: A directed graph G
Output: Layering of G, $L_1, \ldots, L_h$
$S :=$ sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i{+}{+}$
  $G := G \setminus S$
  $S :=$ sources of G



Height Optimization

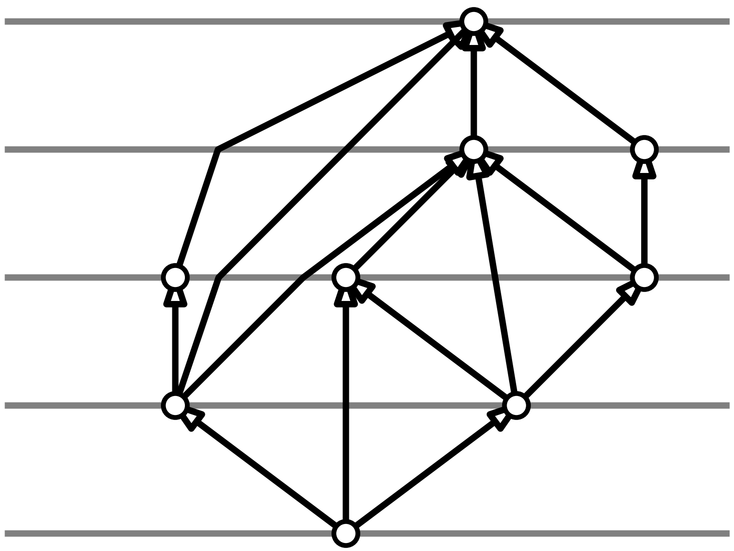
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \dots, L_h$
S := sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i++$
  $G := G \setminus S$
  S := sources of G



Height Optimization

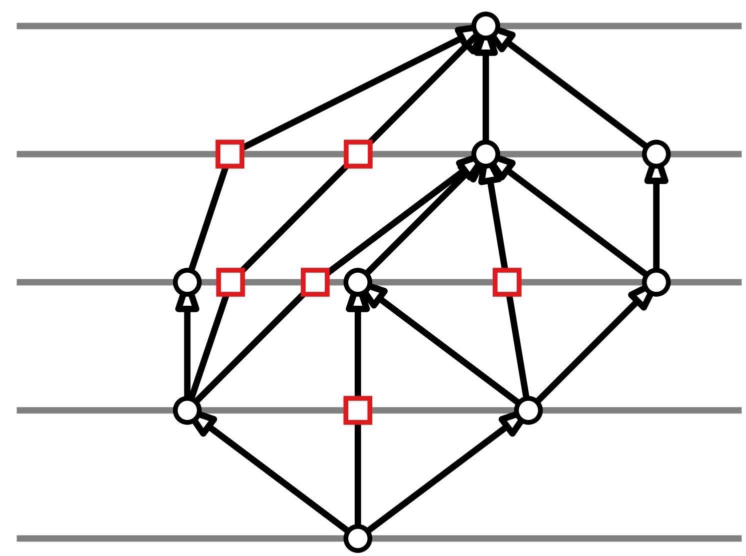
Idea: assign each node v to the layer L_i, where i is the length of the longest simple path from a source to v

- all incomming neighbours lie below v
- the resulting height h is minimized

Algorithm

Input: A directed graph G
Output: Layering of G, $L_1, \dots, L_h$
$S :=$ sources of G
$i = 1$;

while $S \neq \emptyset$ **do**
  $L_i := S$
  $i++$
  $G := G \setminus S$
  $S :=$ sources of G



edges that span more than two layers get subdivided by dummy vertices

Layer Assignement

Other optimization criteria:

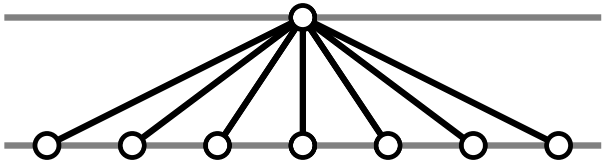
Total edge length: $\sum_{(u,v)\in A}(y(v) - y(u))$ – with integer linear program (polynomial time) [Gansner et al 93]

Layer Assignement

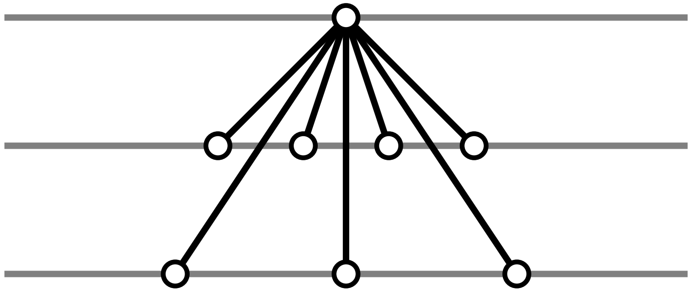
Other optimization criteria:

Total edge length: $\sum_{(u,v)\in A}(y(v) - y(u))$ – with integer linear program (polynomial time) [Gansner et al 93]

Width of the layout:



vs

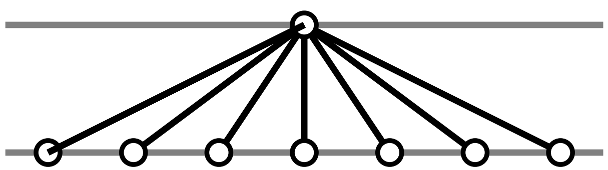


Layer Assignement

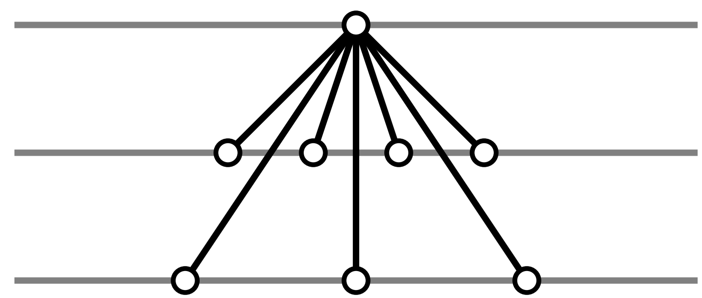
Other optimization criteria:

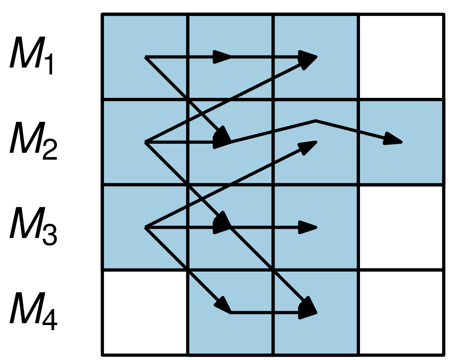
Total edge length: $\sum_{(u,v)\in A}(y(v) - y(u))$ – with integer linear program (polynomial time) [Gansner et al 93]

Width of the layout:



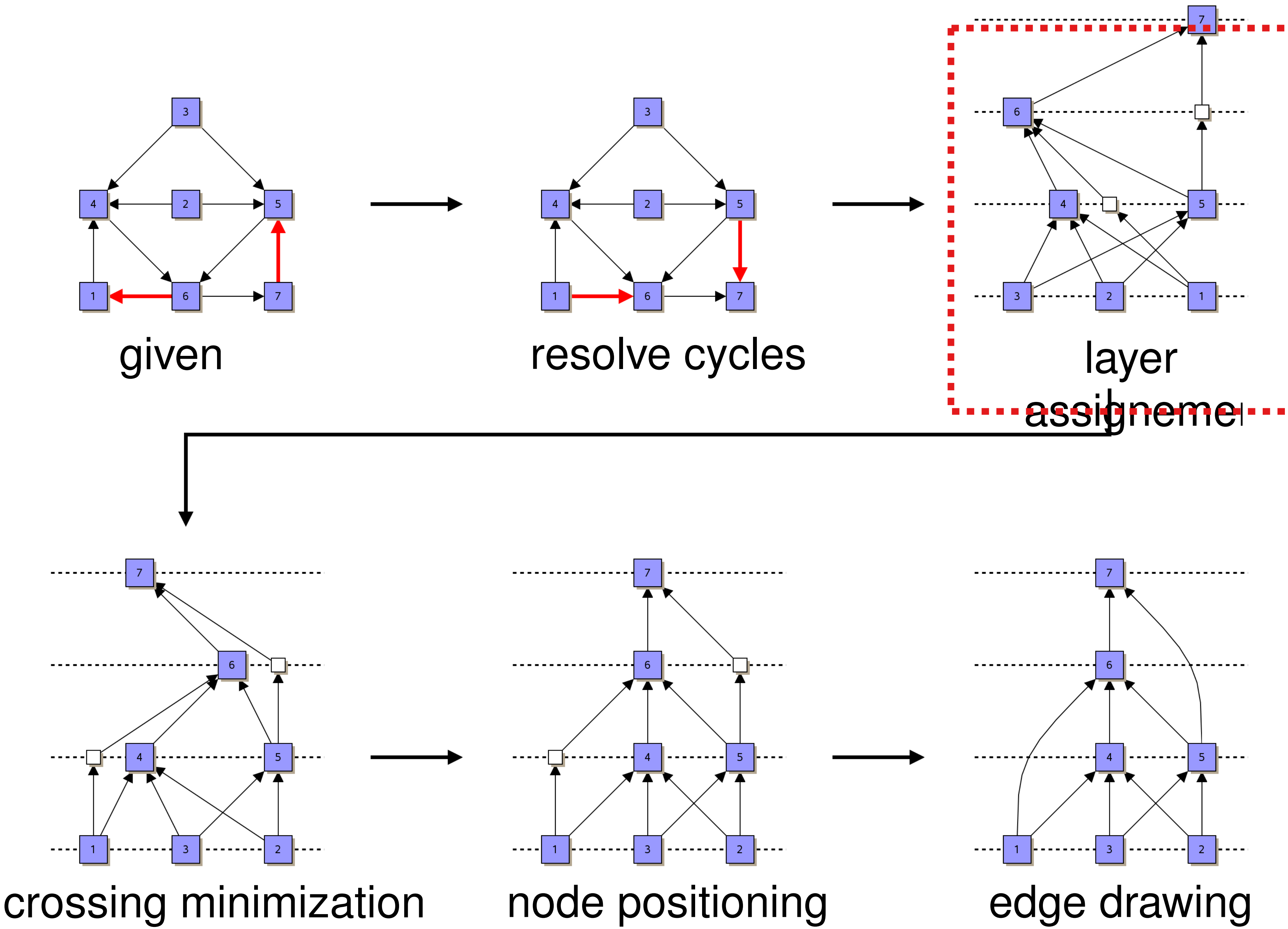
VS



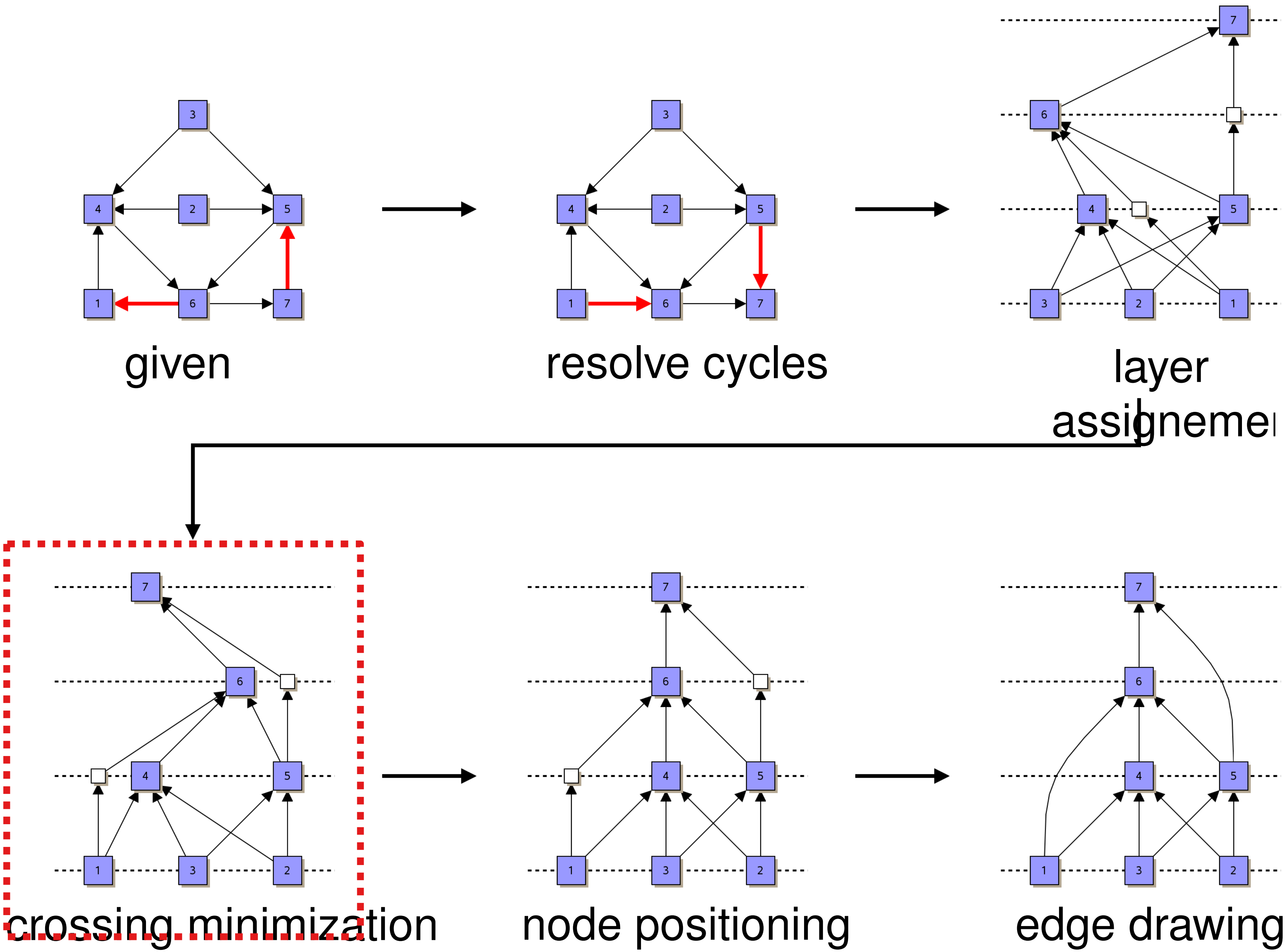


Reduction to scheduling problem – NP-hard but there is a $2 - \frac{1}{B}$-approximation algorithm

Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Problem Statement

Given: DAG $D = (V, A)$, nodes are partitioned in disjoint layers

Find: Order of the nodes on each layer, so that the number of crossing is minimized

Problem Statement

Given: DAG $D = (V, A)$, nodes are partitioned in disjoint layers

Find: Order of the nodes on each layer, so that the number of crossing is minimized

- Number of crossings only depends on the order and not on exact coordinates
- Problem is NP-hard even for two layers (BIPARTITE CROSSING NUMBER [Garey, Johnson '83])
- No approach over several layers simultaneously
- Usually iterative optimization based on approaches for two adjacent layers (*one-sided crossing minimization*)

One-sided Crossing Minimization (OSCM)

Given: 2-Layered-Graph $G = (L_1, L_2, E)$ and ordering of the nodes O_1 of L_1

Find: Node ordering O_2 of L_2, such that the number of crossing among E is minumum

One-sided Crossing Minimization (OSCM)

Given: 2-Layered-Graph $G = (L_1, L_2, E)$ and ordering of the nodes O_1 of L_1

Find: Node ordering O_2 of L_2, such that the number of crossing among E is minumum

Observation:

- for $u, v \in L_2$ the number of crossing among incident to them edges depends on whether $O_2(u) < O_2(v)$ or $O_2(v) < O_2(u)$ and not on the positions of other vertices

One-sided Crossing Minimization (OSCM)

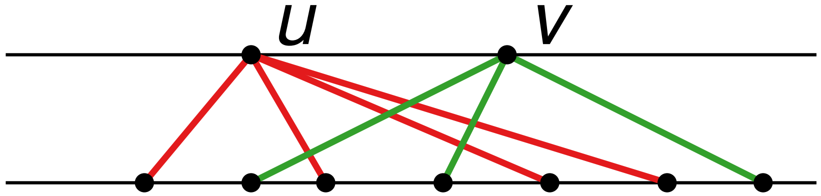
Given: 2-Layered-Graph $G = (L_1, L_2, E)$ and ordering of the nodes O_1 of L_1

Find: Node ordering O_2 of L_2, such that the number of crossing among E is minumum

Observation:

- for $u, v \in L_2$ the number of crossing among incident to them edges depends on whether $O_2(u) < O_2(v)$ or $O_2(v) < O_2(u)$ and not on the positions of other vertices

Def: $c_{uv} := |\{(uw, vz) : w \in N(u), z \in N(v), O_1(z) < O_1(w)\}|$ for $O_2(u) < O_2(v)$



$c_{uv} = 5$
$c_{vu} = 7$

One-sided Crossing Minimization (OSCM)

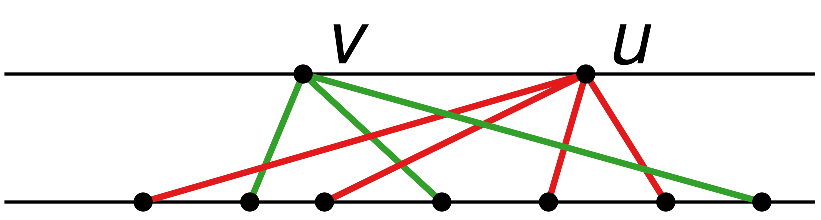
Given: 2-Layered-Graph $G = (L_1, L_2, E)$ and ordering of the nodes O_1 of L_1

Find: Node ordering O_2 of L_2, such that the number of crossing among E is minumum

Observation:

- for $u, v \in L_2$ the number of crossing among incident to them edges depends on whether $O_2(u) < O_2(v)$ or $O_2(v) < O_2(u)$ and not on the positions of other vertices

Def: $c_{uv} := |\{(uw, vz) : w \in N(u), z \in N(v), O_1(z) < O_1(w)\}|$ for $O_2(u) < O_2(v)$



$c_{uv} = 5$
$c_{vu} = 7$

Further Properties

Def: Crossing number of G with orders x_1 and x_2 for L_1 and L_2 is denoted by $\mathrm{cr}(G, O_1, O_2)$;
for fixed O_1 then $\mathrm{opt}(G, O_1) = \min_{O_2} \mathrm{cr}(G, O_1, O_2)$

- It holds that $\mathrm{cr}(G, O_1, O_2) = \sum_{O_2(u) < O_2(v)} c_{uv}$ - gives a way to compute $\mathrm{cr}(G, O_1, O_2)$ in $O(m^2)$, where m-number of edges

Further Properties

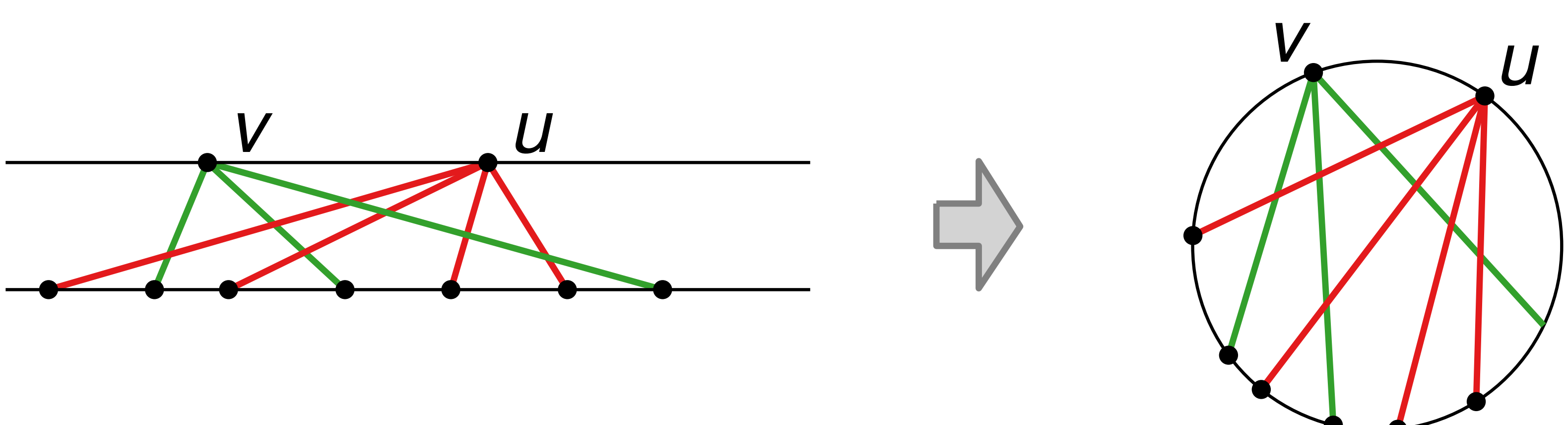
Def: Crossing number of G with orders x_1 and x_2 for L_1 and L_2 is denoted by $\mathrm{cr}(G, O_1, O_2)$;
for fixed O_1 then $\mathrm{opt}(G, O_1) = \min_{O_2} \mathrm{cr}(G, O_1, O_2)$

- It holds that $\mathrm{cr}(G, O_1, O_2) = \sum_{O_2(u) < O_2(v)} c_{uv}$ - gives a way to compute $\mathrm{cr}(G, O_1, O_2)$ in $O(m^2)$, where m-number of edges
- $\mathrm{cr}(G, O_1, O_2)$ can be computed in $O(m + C)$ time, where C-number of crossings [Six and Tollis 2006] $\rightarrow$ reduce to counting crossings in a circular drawing

Further Properties

Def: Crossing number of G with orders x_1 and x_2 for L_1 and L_2 is denoted by $\mathrm{cr}(G, O_1, O_2)$;
for fixed O_1 then $\mathrm{opt}(G, O_1) = \min_{O_2} \mathrm{cr}(G, O_1, O_2)$

- It holds that $\mathrm{cr}(G, O_1, O_2) = \sum_{O_2(u)<O_2(v)} c_{uv}$ - gives a way to compute $\mathrm{cr}(G, O_1, O_2)$ in $O(m^2)$, where m-number of edges
- $\mathrm{cr}(G, O_1, O_2)$ can be computed in $O(m + C)$ time, where C-number of crossings [Six and Tollis 2006] $\rightarrow$ reduce to counting crossings in a circular drawing



Iterative Crossing Minimization

Let $G = (V, E)$ be a DAG with layers $L_1, \dots, L_h$.

(1) compute a random ordering O_1 for layer L_1

(2) for $i = 1, \dots, h-1$ consider layers L_i and L_{i+1} and minimize $\mathrm{cr}(G, O_i, O_{i+1})$ with fixed O_i ($\rightarrow$ **OSCM**)

(3) for $i = h-1, \dots, 1$ consider layers L_{i+1} and L_i and minimize $\mathrm{cr}(G, O_i, O_{i+1})$ with fixed O_{i+1} ($\rightarrow$ **OSCM**)

(4) repeat (2) and (3) until no further improvement happens

(5) repeat steps (1)–(4) with another O_1

(6) return the best found solution

Iterative Crossing Minimization

Let $G = (V, E)$ be a DAG with layers $L_1, \dots, L_h$.

(1) compute a random ordering O_1 for layer L_1
(2) for $i = 1, \dots, h-1$ consider layers L_i and L_{i+1} and minimize $\mathrm{cr}(G, O_i, O_{i+1})$ with fixed O_i ($\rightarrow$ **OSCM**)
(3) for $i = h-1, \dots, 1$ consider layers L_{i+1} and L_i and minimize $\mathrm{cr}(G, O_i, O_{i+1})$ with fixed O_{i+1} ($\rightarrow$ **OSCM**)
(4) repeat (2) and (3) until no further improvement happens
(5) repeat steps (1)–(4) with another O_1
(6) return the best found solution

Theorem 1: The One-Sided Crossing Minimization (OSCM) problem is NP-hard [Eades, Wormald 1994].

Algorithms for OSCM

Heuristics:

- Barycenter [Sugiyama et al, 81]
- Median [Eades and Wormald, 94]

Exact:

- ILP Model [Juenger and Mutzel, 97]

...and many more...

Barycenter Heuristic (Sugiyama, Tagawa, Toda 1981)

Idea: few crossing when nodes are close to their neighbours

- set

$$o_2(u) = \frac{1}{\deg(u)} \sum_{v \in N(u)} o_1(v)$$

- in case of equality introduce tiny gap

Barycenter Heuristic (Sugiyama, Tagawa, Toda 1981)

Idea: few crossing when nodes are close to their neighbours

- set

$$o_2(u) = \frac{1}{\deg(u)} \sum_{v \in N(u)} o_1(v)$$

- in case of equality introduce tiny gap

Properties:

- trivial implementation
- quick (exactly?)
- usually very good results
- finds optimum if $\mathrm{opt}(G, o_1) = 0$
- there are graphs on which it performs $\Omega(\sqrt{n})$ times worse than optimal

Barycenter Heuristic (Sugiyama, Tagawa, Toda 1981)

Idea: few crossing when nodes are close to their neighbours

- set

$$o_2(u) = \frac{1}{\deg(u)} \sum_{v \in N(u)} o_1(v)$$

- in case of equality introduce tiny gap

Properties:

- trivial implementation
- quick (exactly?)
- usually very good results
- finds optimum if $\mathrm{opt}(G, o_1) = 0$
- there are graphs on which it performs $\Omega(\sqrt{n})$ times worse than optimal

Median-Heuristic (Eades, Wormald 1994)

Idea: use the median of the coordinates of neighbours

- for a node $v \in L_2$ with neighbours $v_1, \ldots, v_k$ set $o_2(v) = \text{med}(v) = o_1(v_{\lceil k/2 \rceil})$ and $o_2(v) = 0$ if $N(v) = \emptyset$
- if $o_2(u) = o_2(v)$ and u, v have different degree parity, place the node with odd degree to the left
- if $o_2(u) = o_2(v)$ and u, v have the same degree parity, place an arbitrary of them to the left
- Runs in time $O(|E|)$

Median-Heuristic (Eades, Wormald 1994)

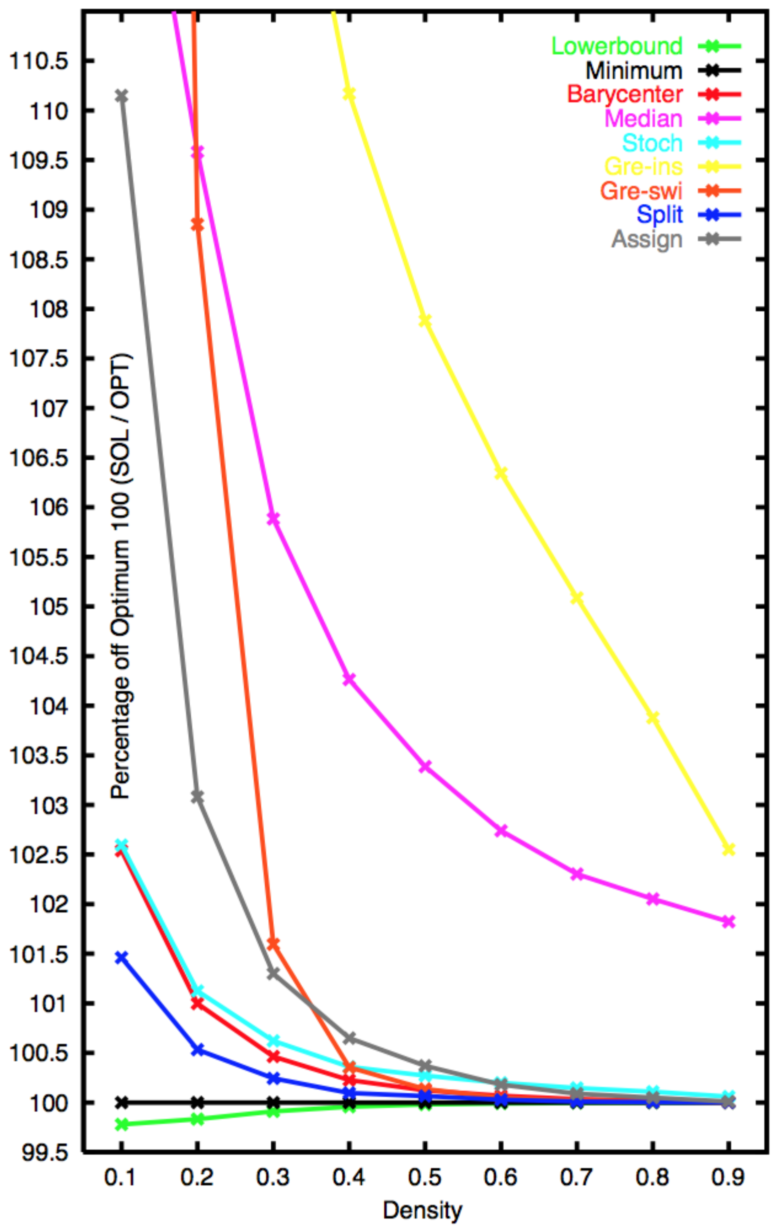
Idea: use the median of the coordinates of neighbours

- for a node $v \in L_2$ with neighbours $v_1, \dots, v_k$ set $o_2(v) = \text{med}(v) = o_1(v_{\lceil k/2 \rceil})$ and $o_2(v) = 0$ if $N(v) = \emptyset$
- if $o_2(u) = o_2(v)$ and u, v have different degree parity, place the node with odd degree to the left
- if $o_2(u) = o_2(v)$ and u, v have the same degree parity, place an arbitrary of them to the left
- Runs in time $O(|E|)$

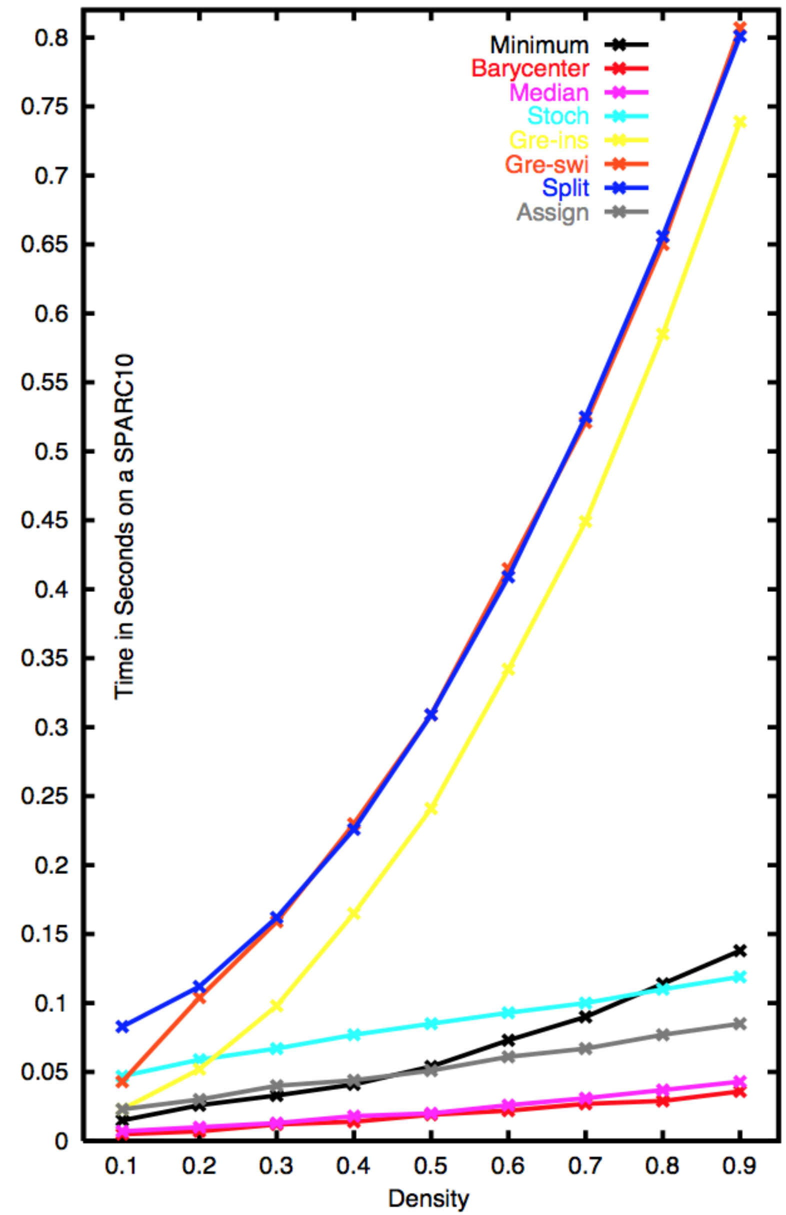
Properties:

- trivial implementation
- fast
- mostly good performance
- finds optimum when $\text{opt}(G, o_1) = 0$
- **Factor-3 Approximation**: $\text{med}(G, o_1) \leq 3\,\text{opt}(G, o_1)$

Experimental Evaluation (Jünger, Mutzel 1997)

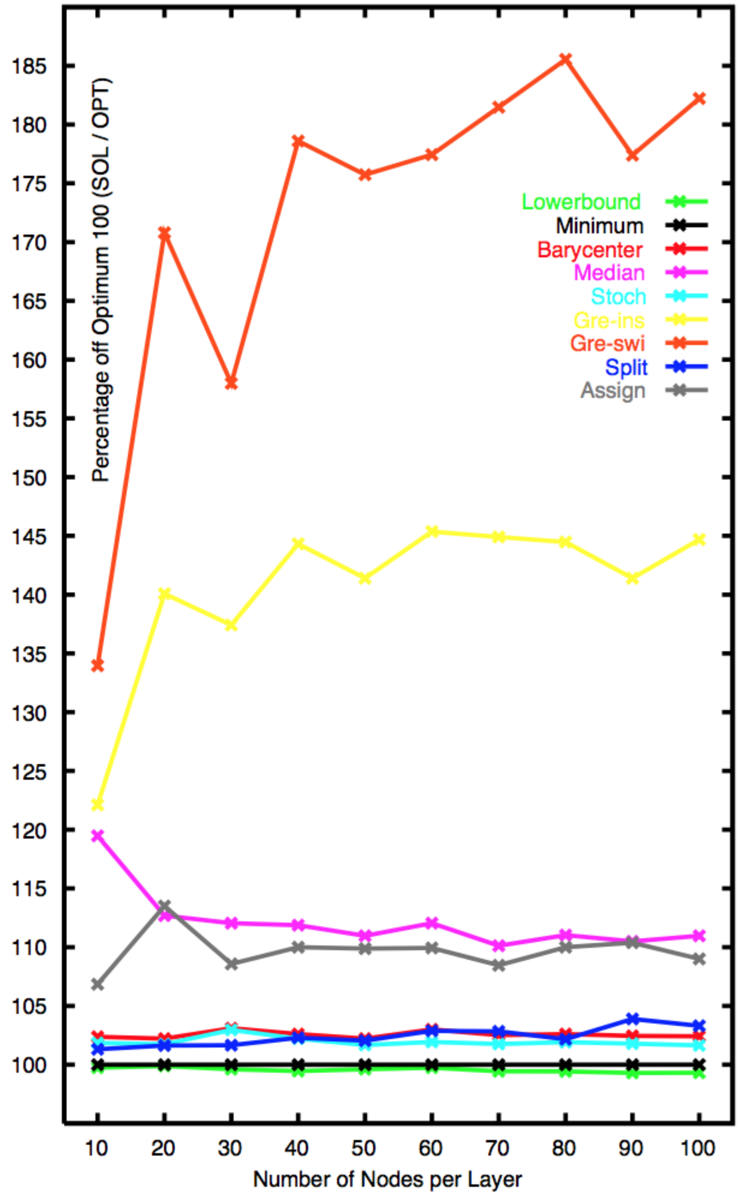


Results for 100 instances on 20 + 20 nodes with increasing density

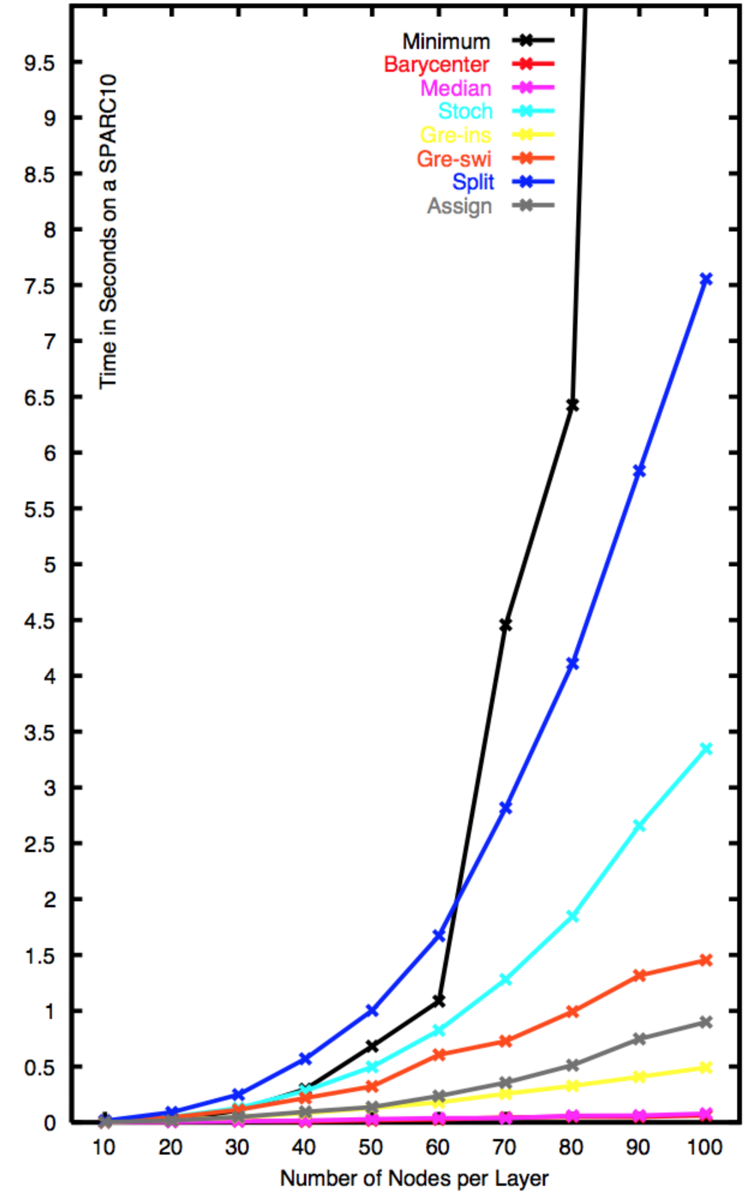


Time for 100 instances on 20 + 20 nodes with increasing density

Experimental Evaluation (Jünger, Mutzel 1997)

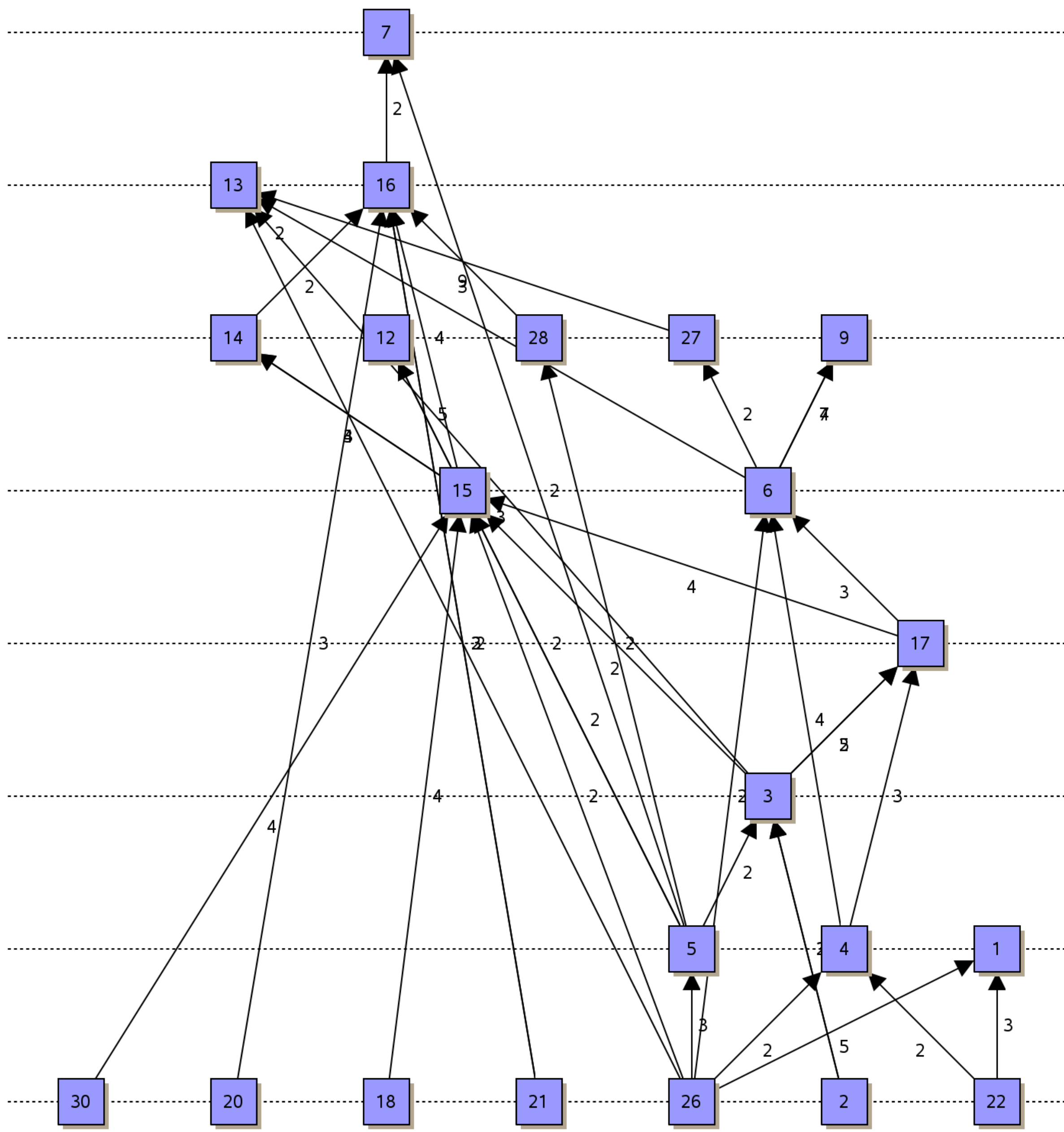


Results for 10 instances of sparse graphs with increasing size

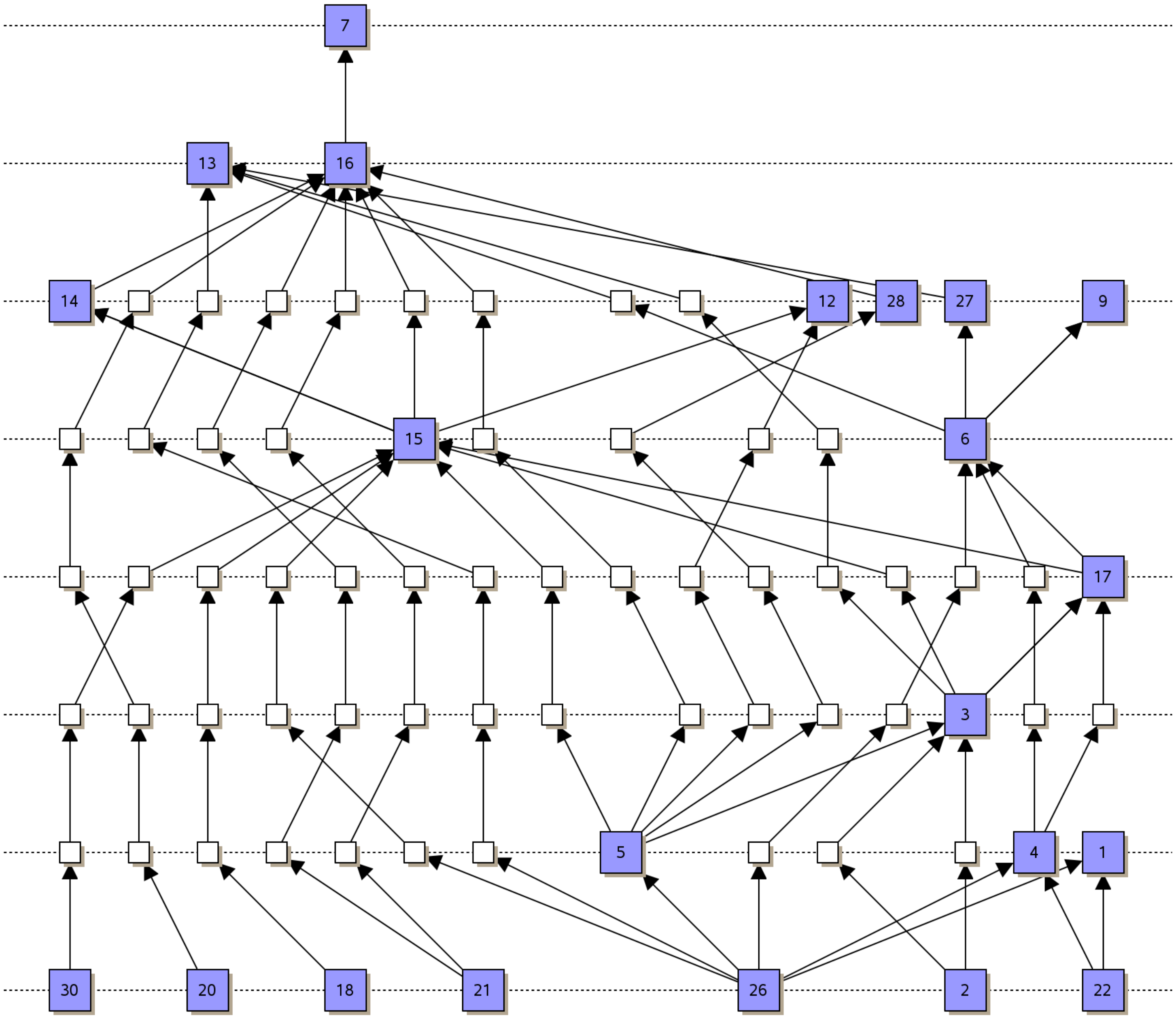


Time for 10 instances of sparse graphs with increasing size

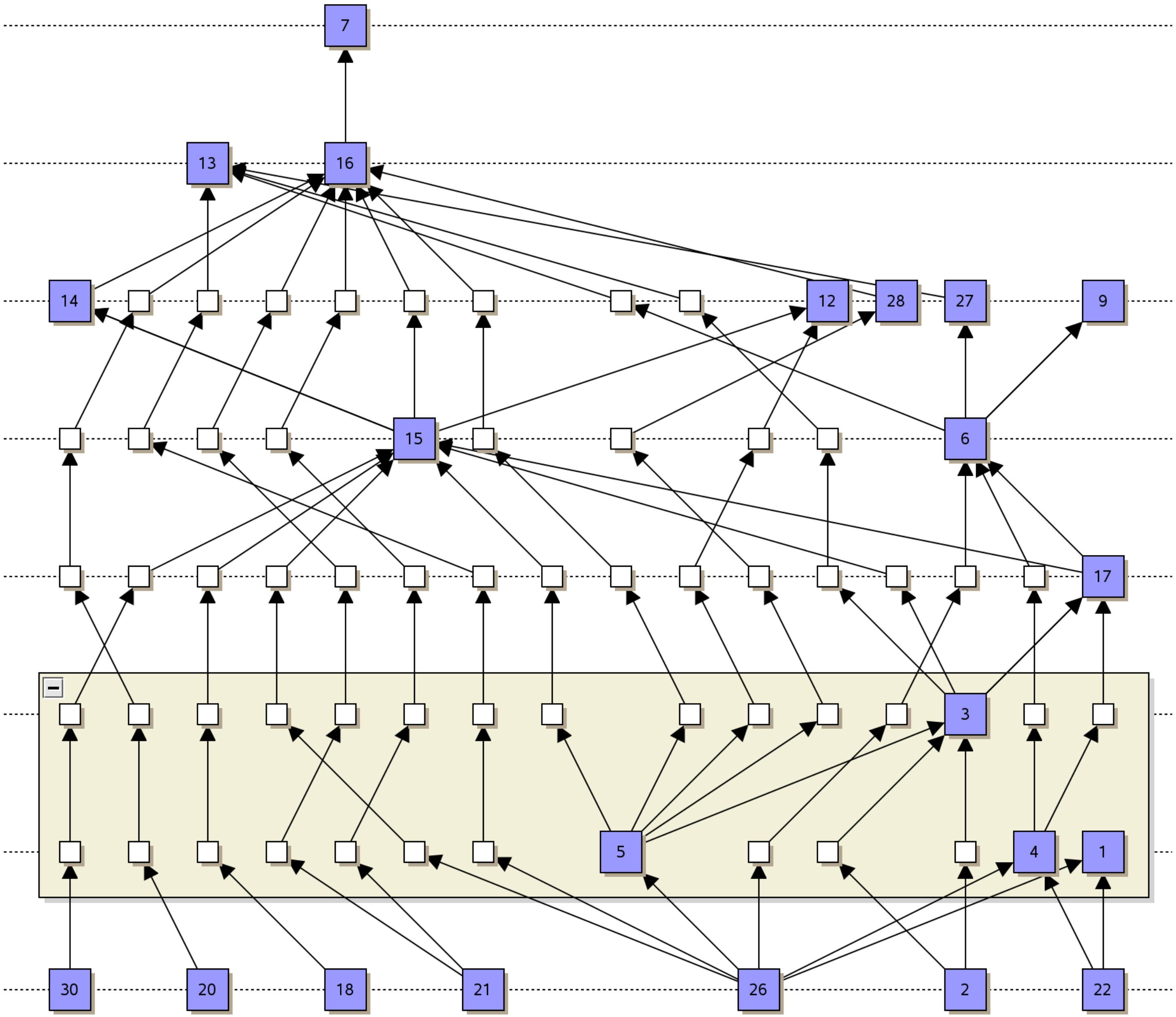
Example with Barycenter



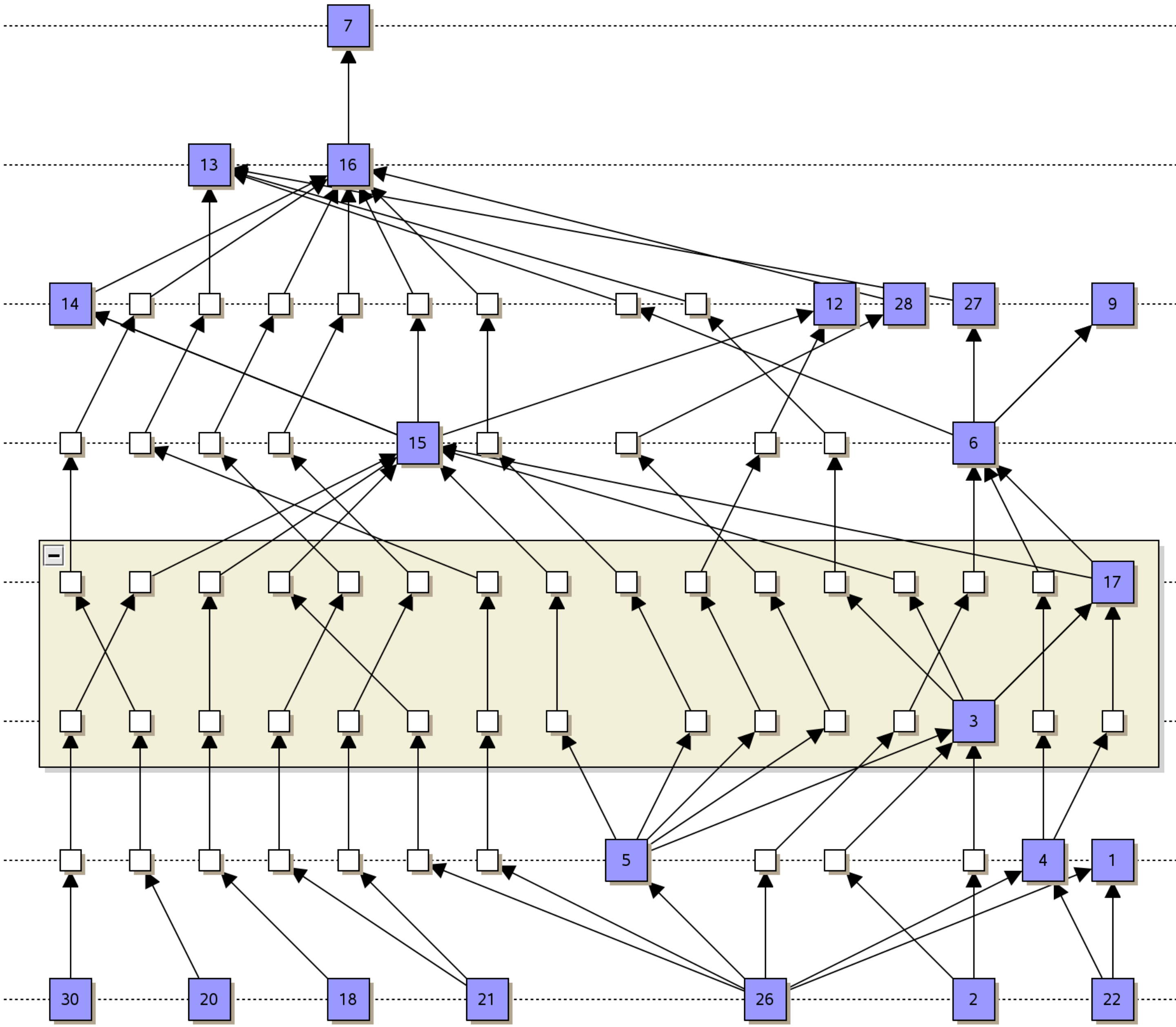
Example with Barycenter



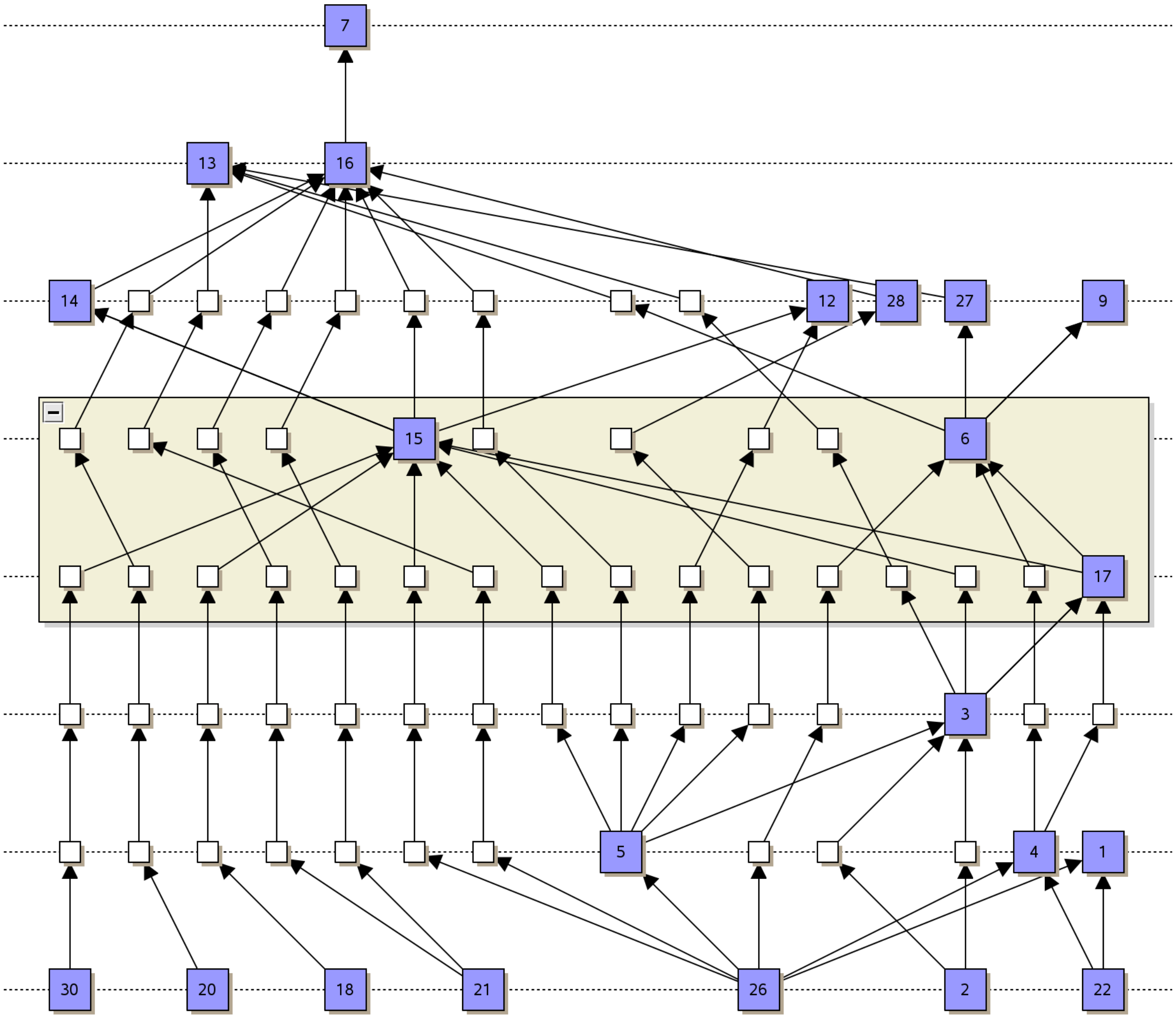
Example with Barycenter



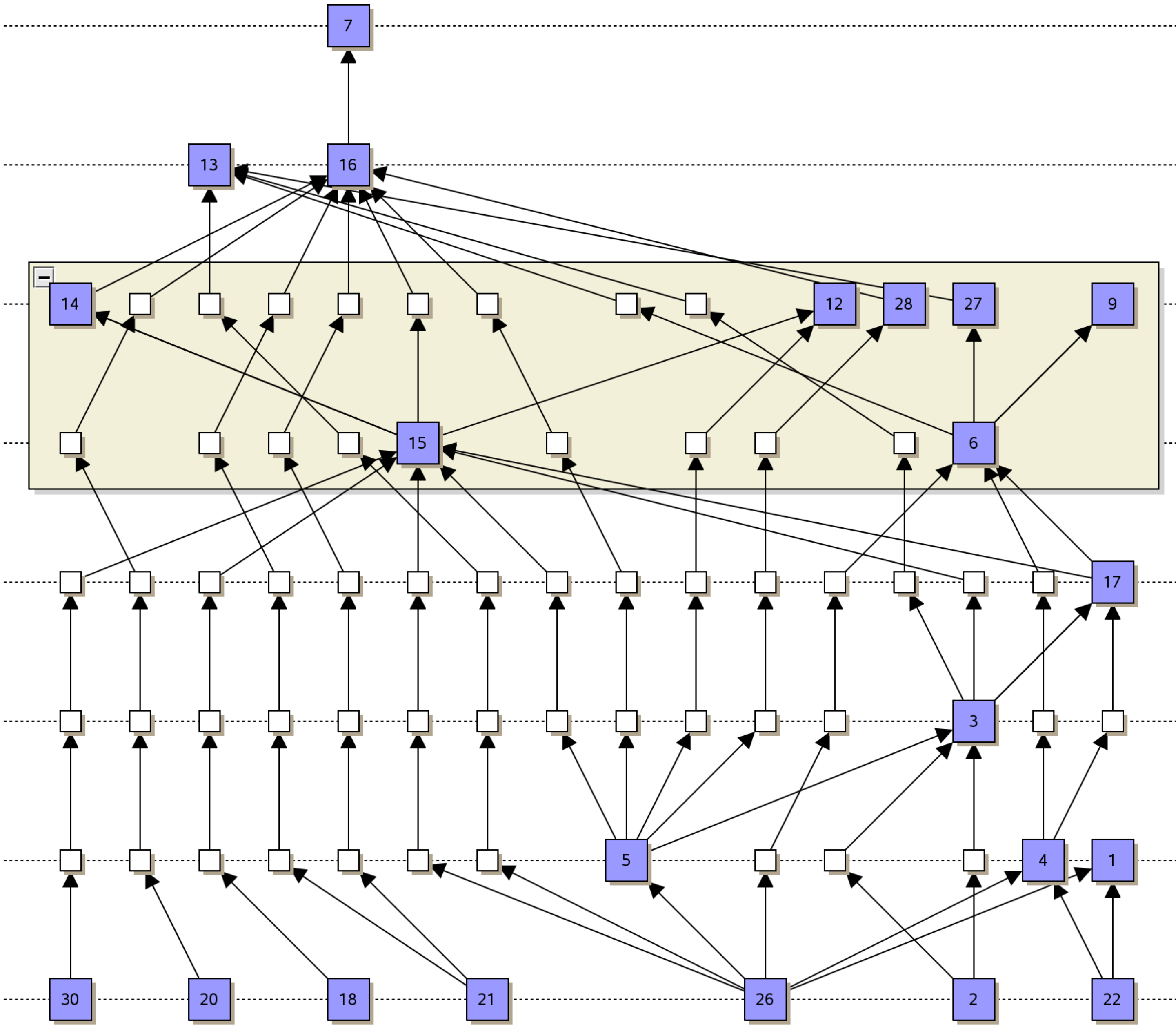
Example with Barycenter



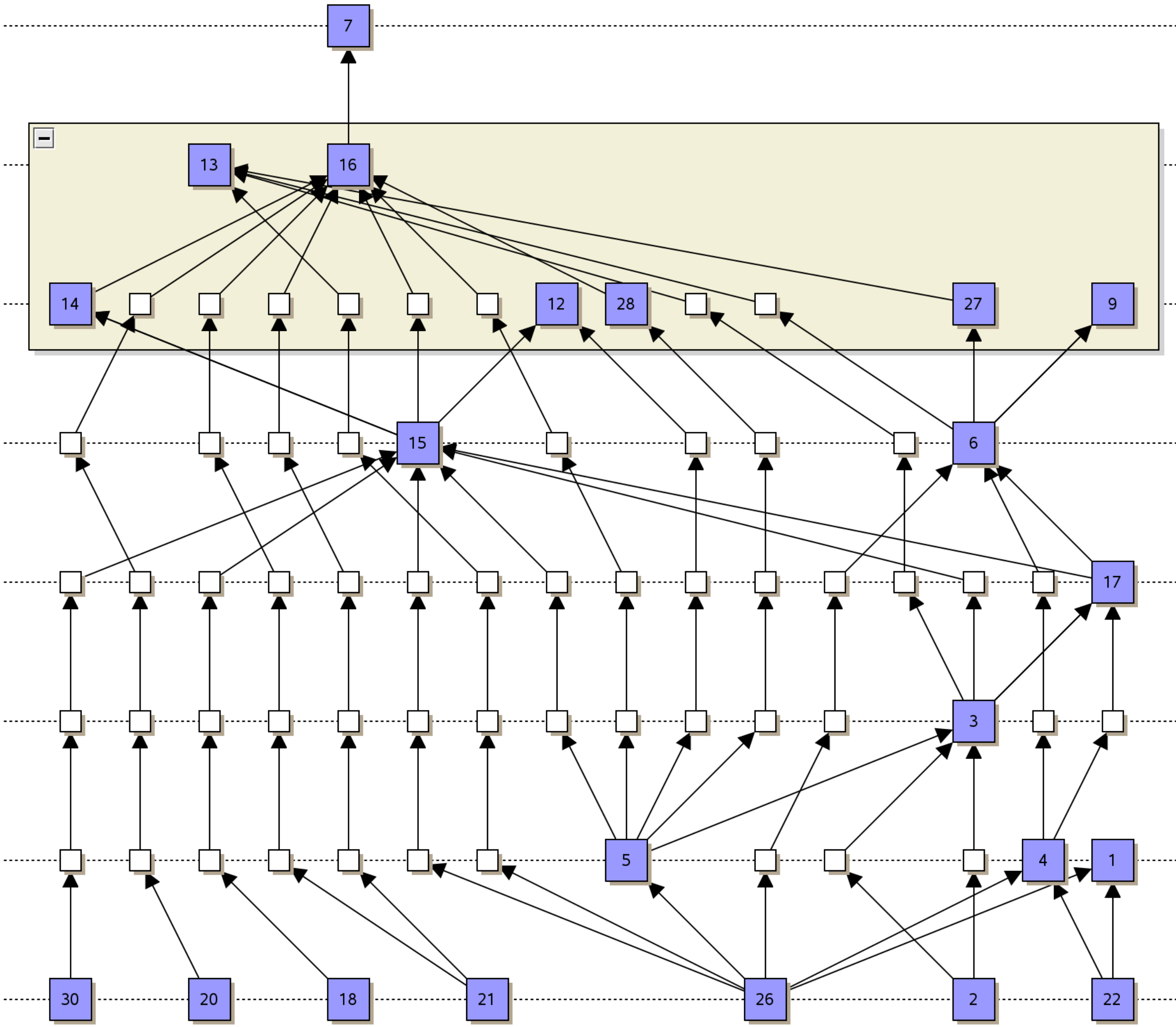
Example with Barycenter



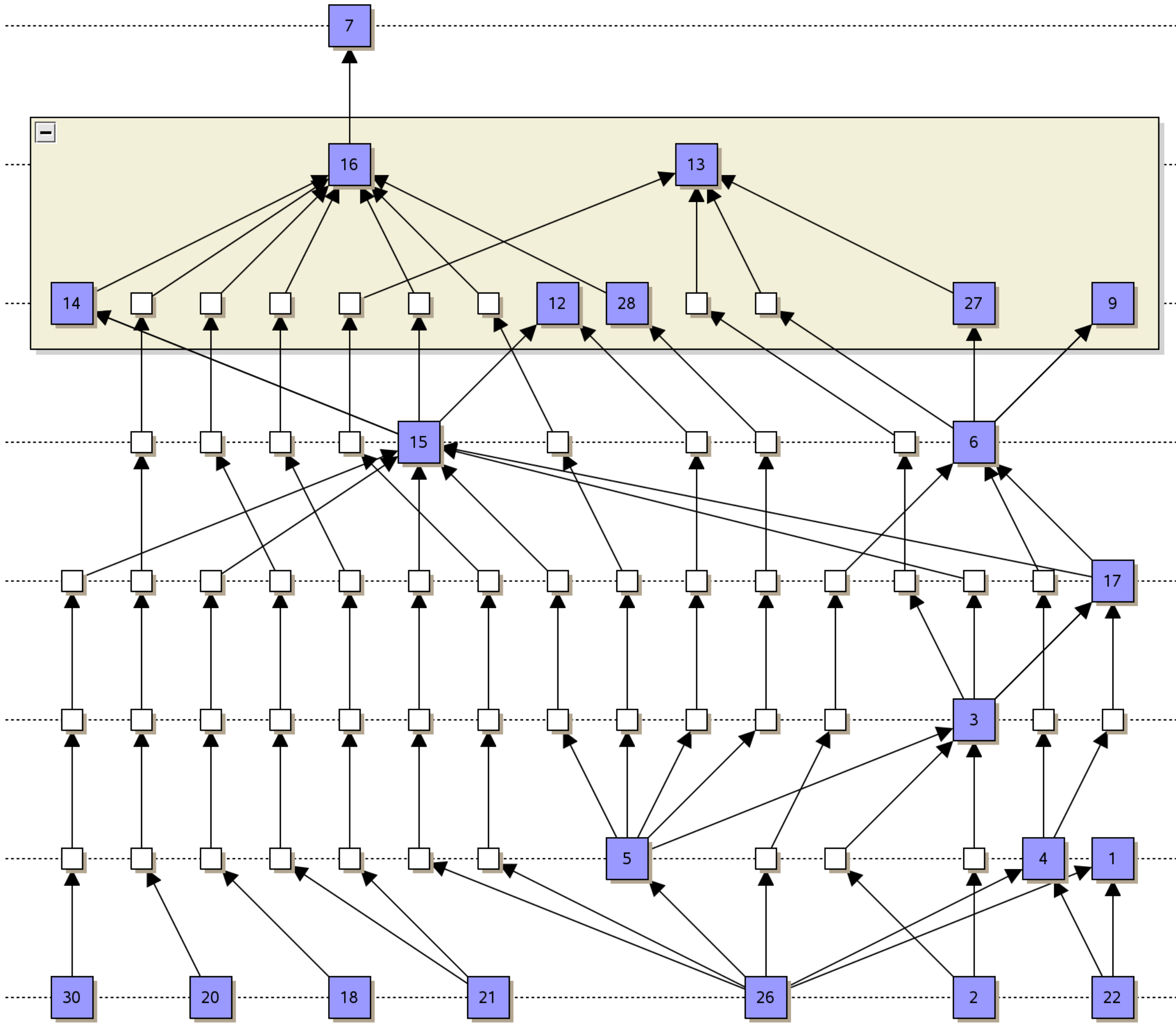
Example with Barycenter



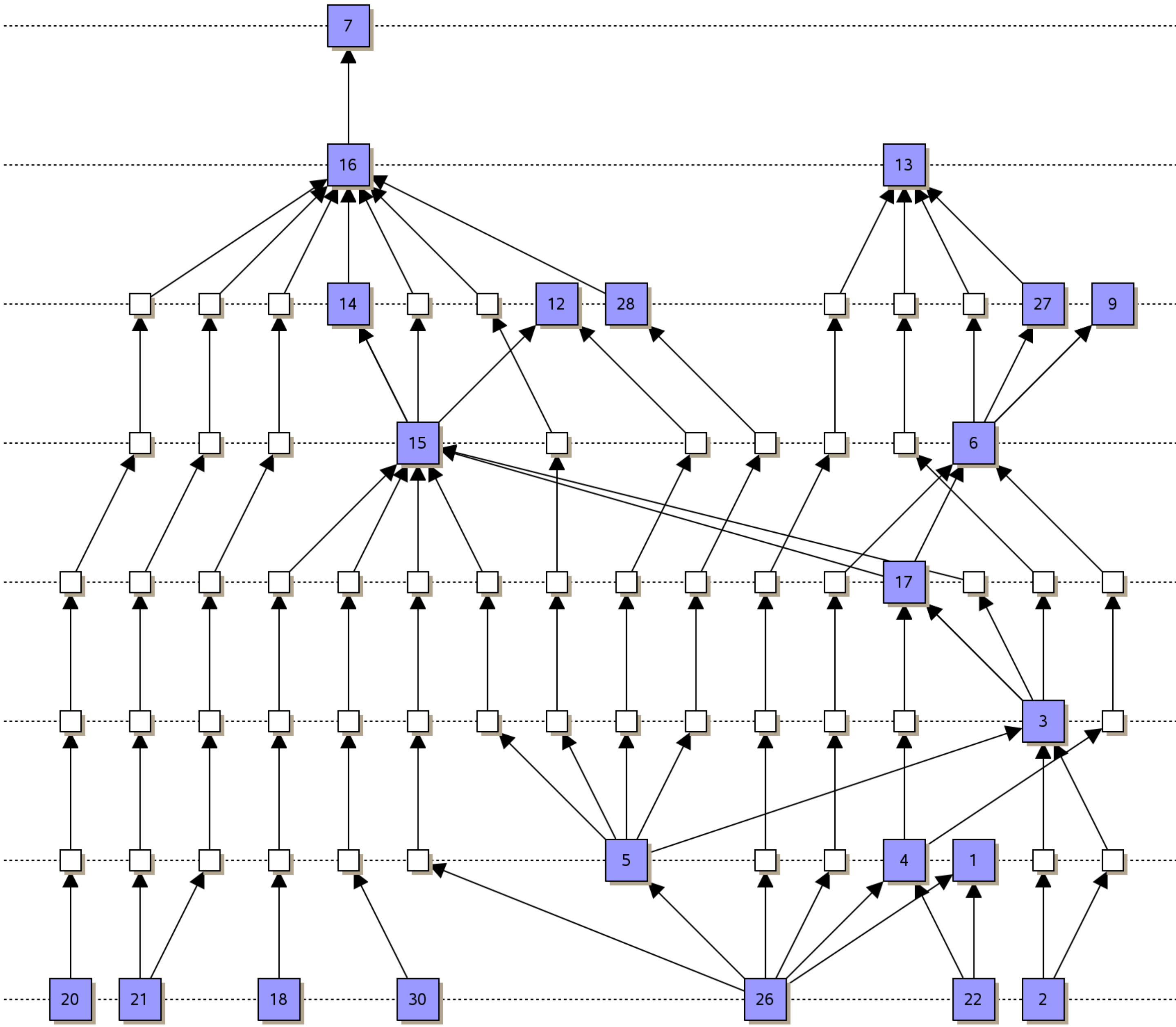
Example with Barycenter



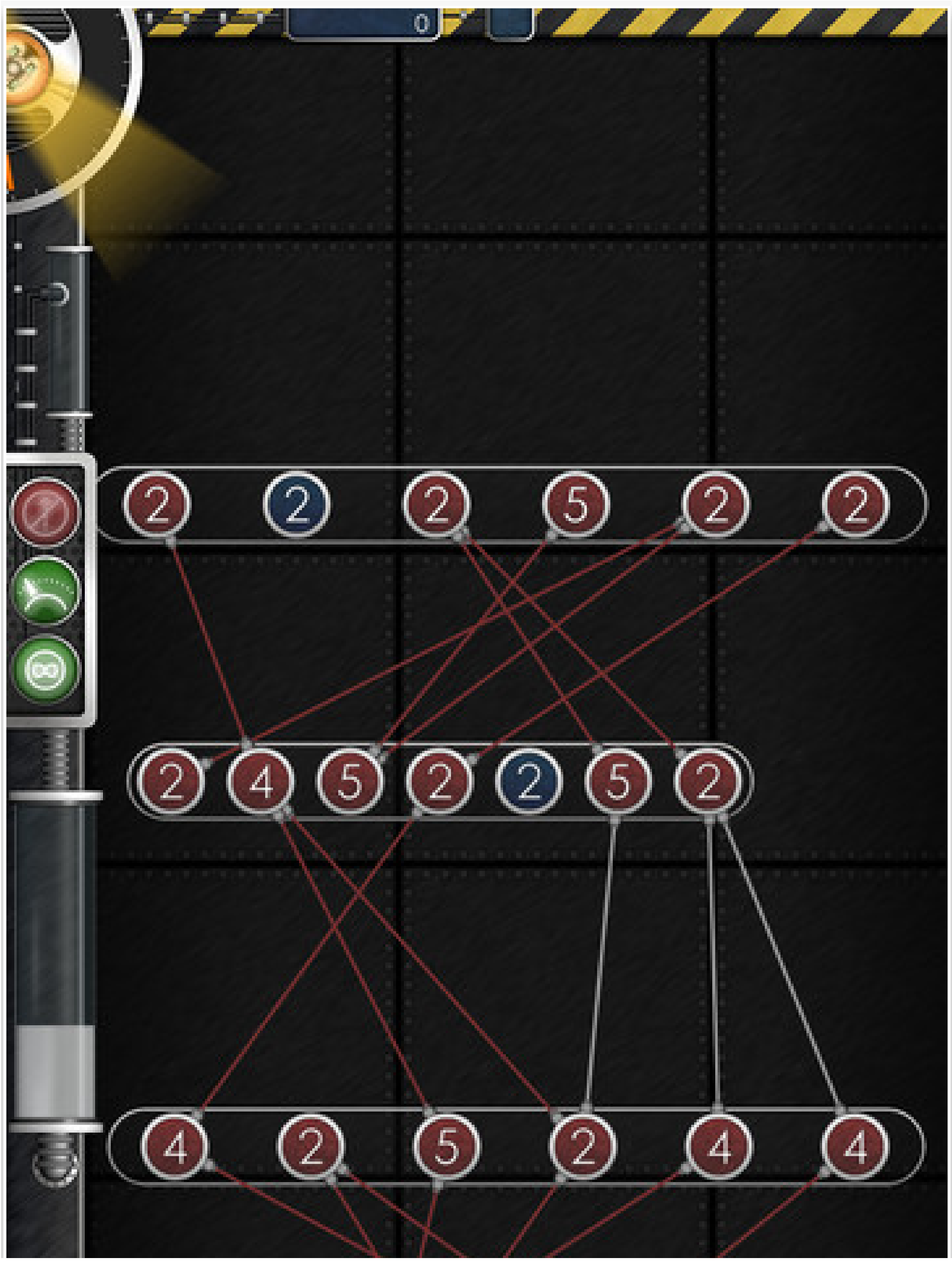
Example with Barycenter



Example with Barycenter



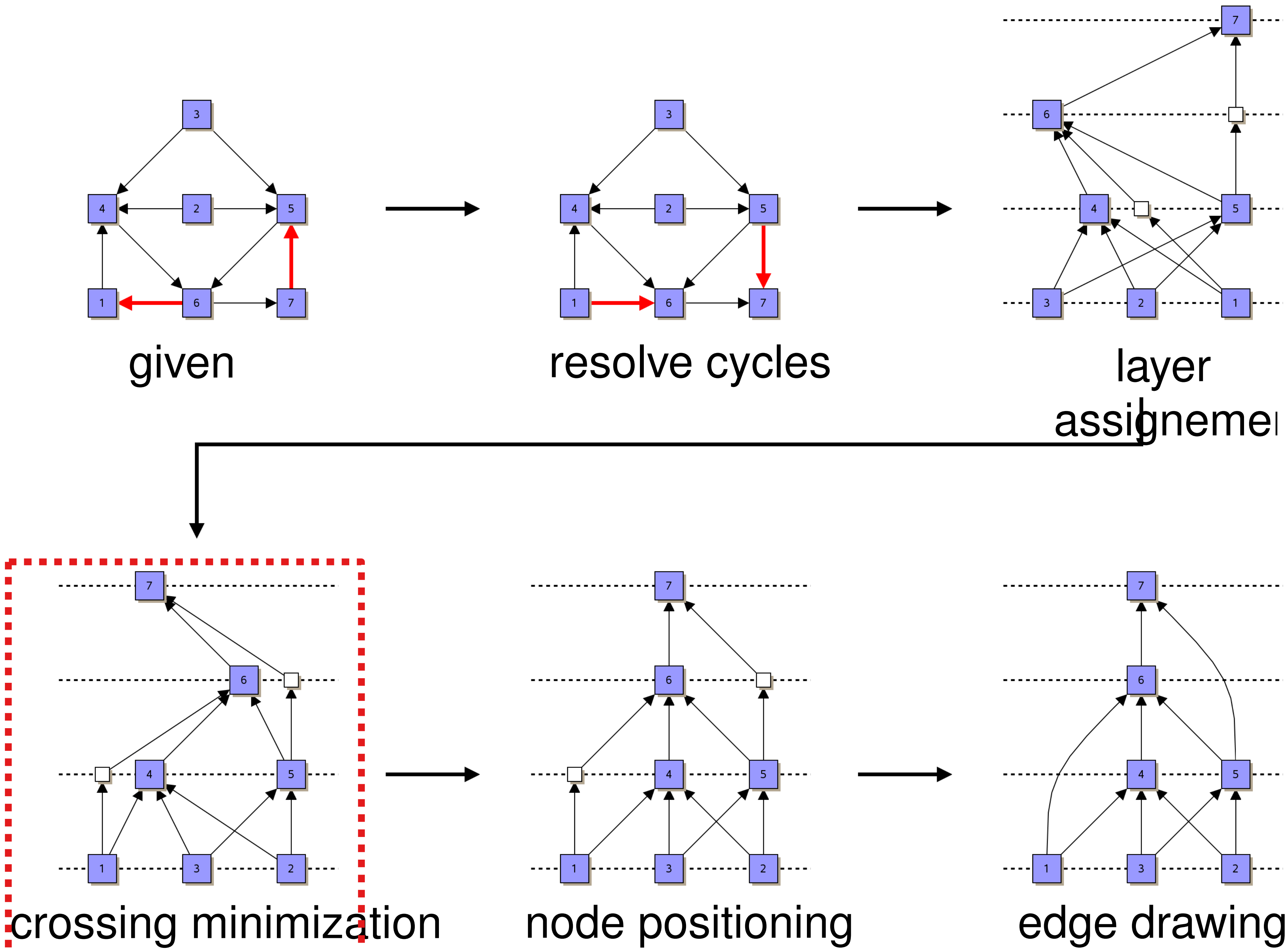
CrossingX



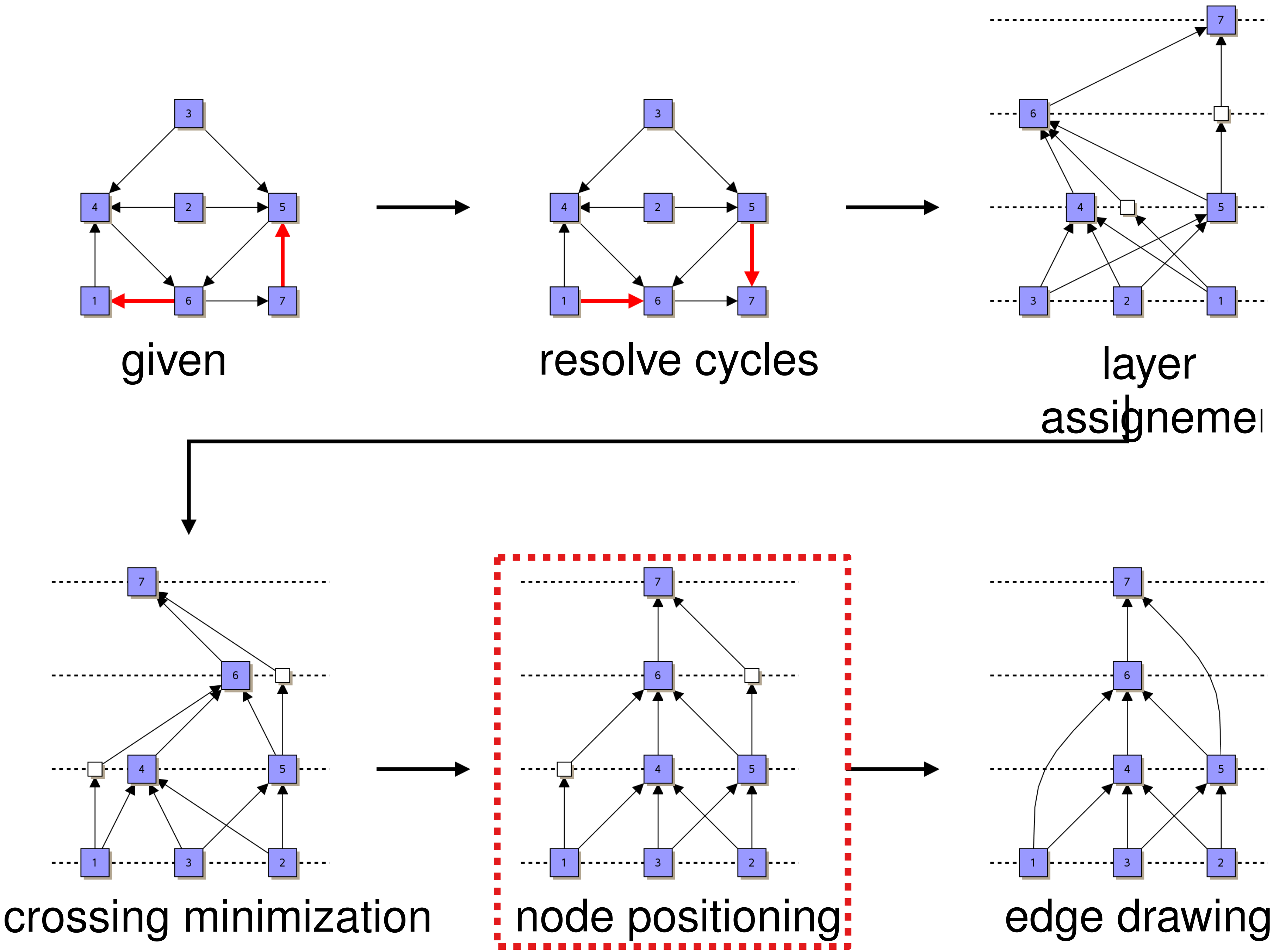
There was even an iPad game **CrossingX** for the OSCM Problem!

Winner of Graph Drawing Game Contest 2012

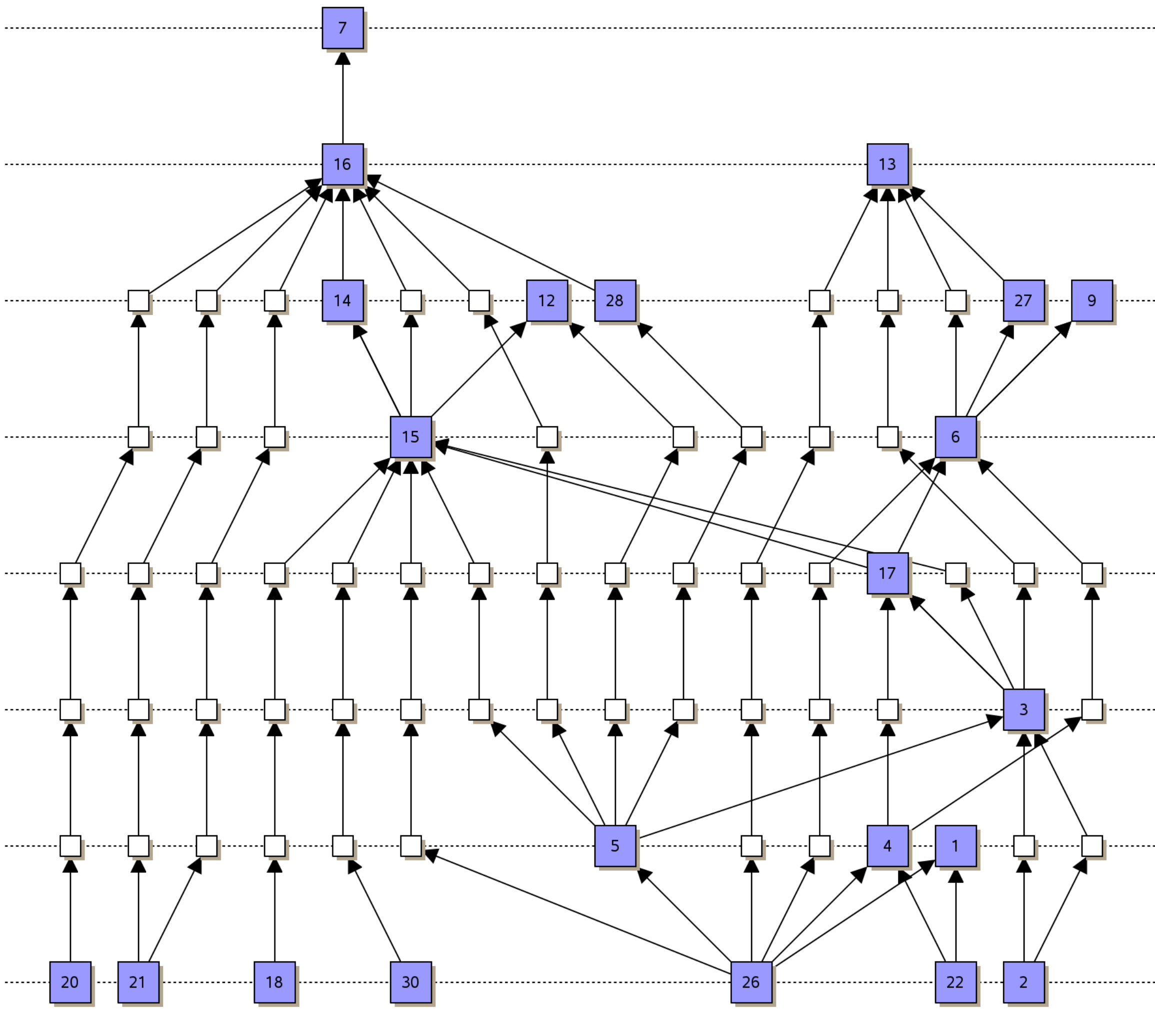
Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



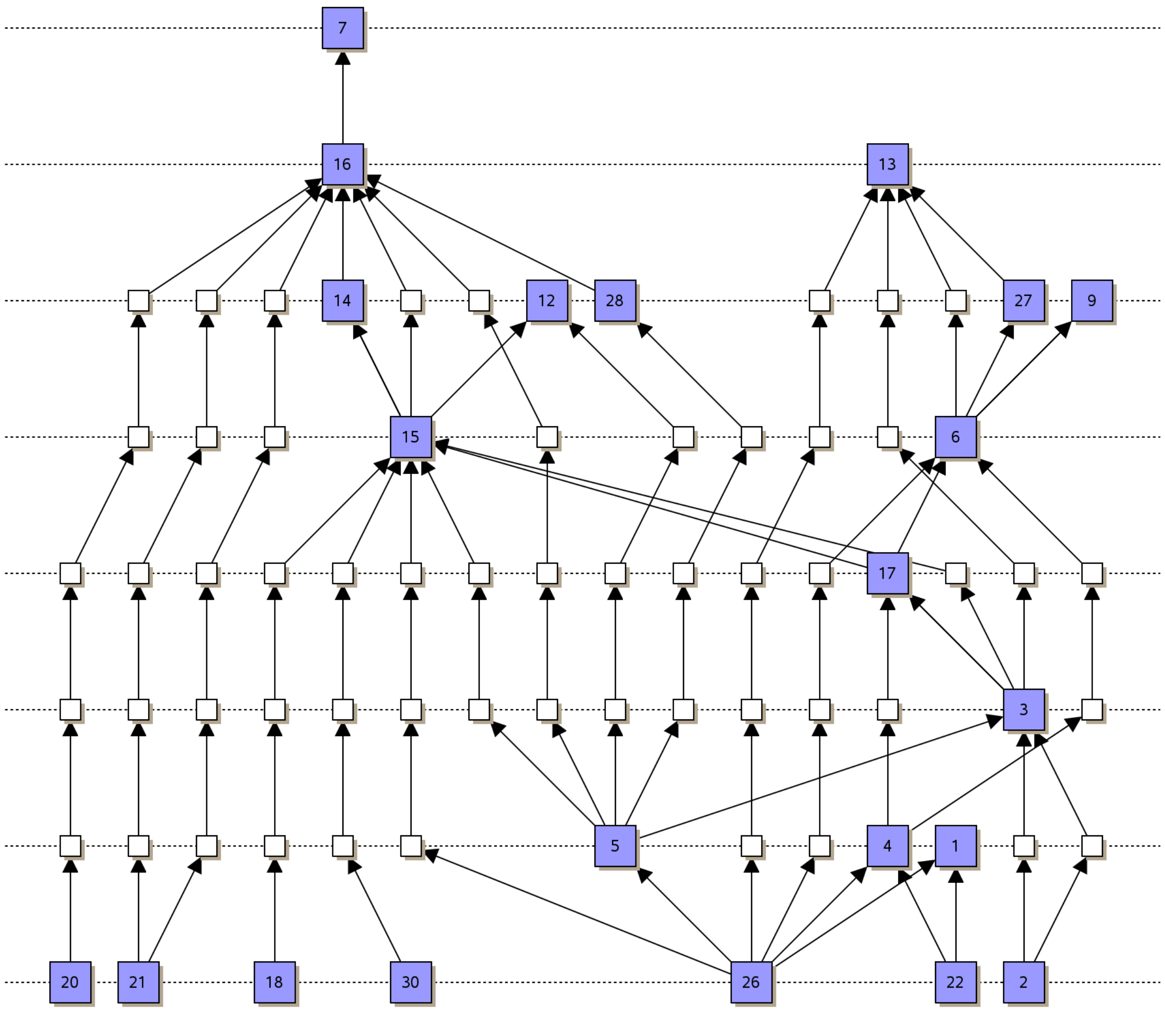
Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



Coordinate Assignement Goals



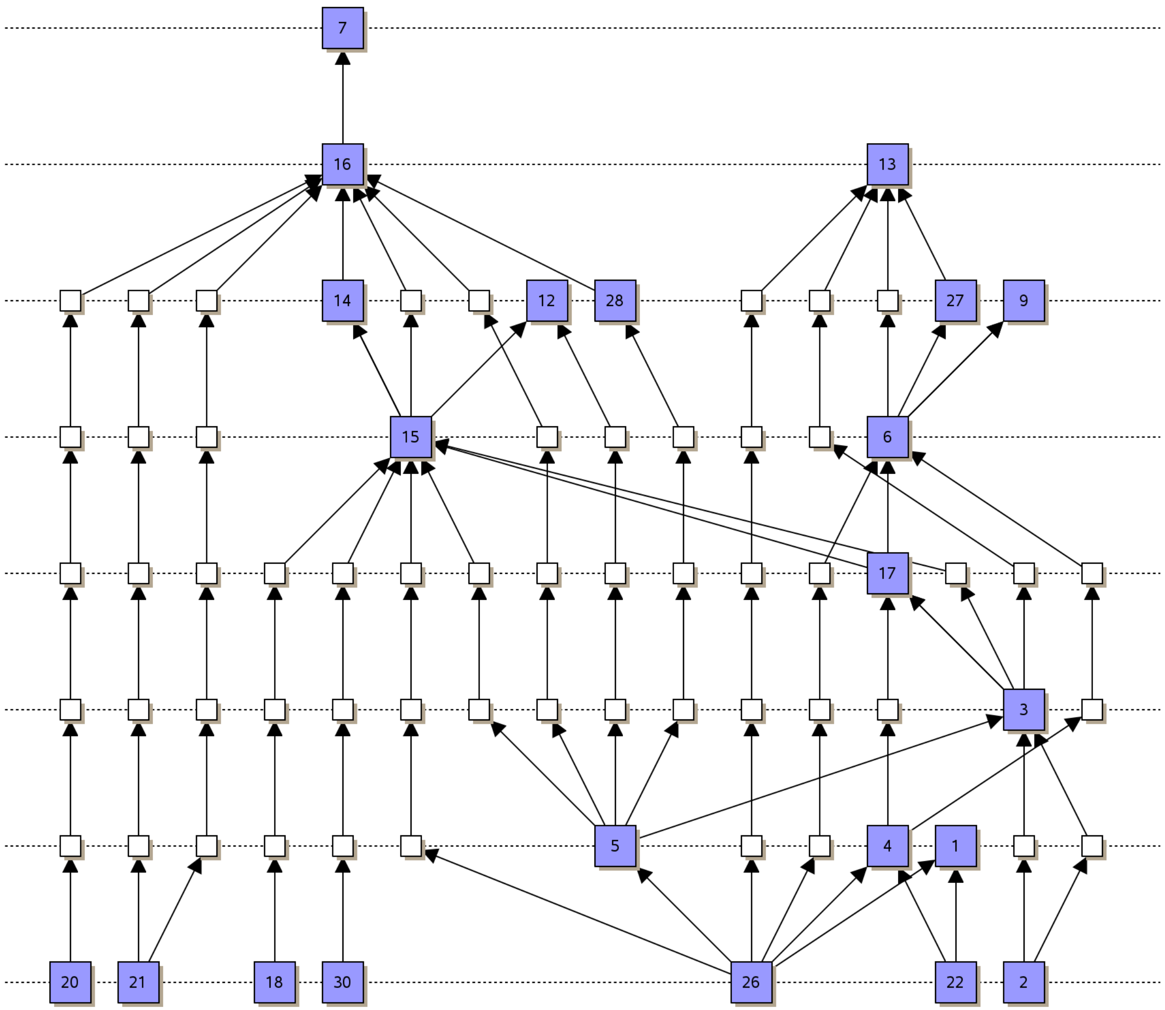
Coordinate Assignement Goals



Think for a minute and then share

What could we optimize when doing the coordinate assignment?

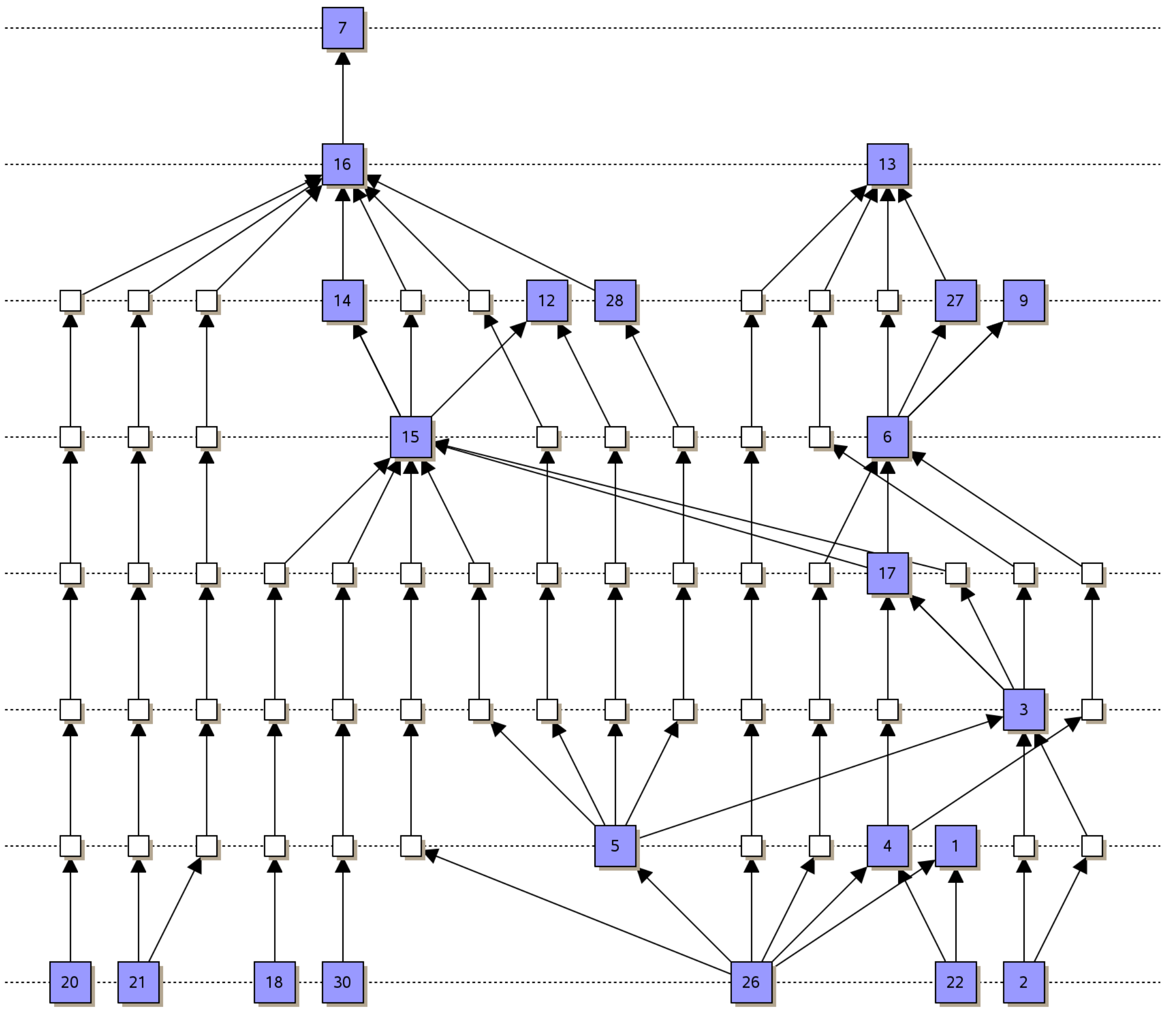
Coordinate Assignement Goals



Think for a minute and then share

What could we optimize when doing the coordinate assignment?

Coordinate Assignement Goals



- Edges as straight as possible
- Edges as vertical as possible

Steightening Edges

Goal: minimize deviation from a straight-line for the edges with dummy-nodes

Idea: use quadratic Program

- let $p_{uv} = (u, d_1, \dots, d_k, v)$ path with k dummy nodes betwen u and v
- let $a_i = x(u) + \frac{i}{k+1}(x(v) - x(u))$ the x-coordinate of d_i when (u, v) is straight
- $g(p_{uv}) = \sum_{i=1}^{k}(x(d_i) - a_i)^2$
- minimize $\sum_{uv \in E} g(p_{uv})$
- constraints: $x(w) - x(z) \geq \delta$ for consecutive nodes on the same layer, w on the right of z (δ distance parameter)

Steightening Edges

Goal: minimize deviation from a straight-line for the edges with dummy-nodes

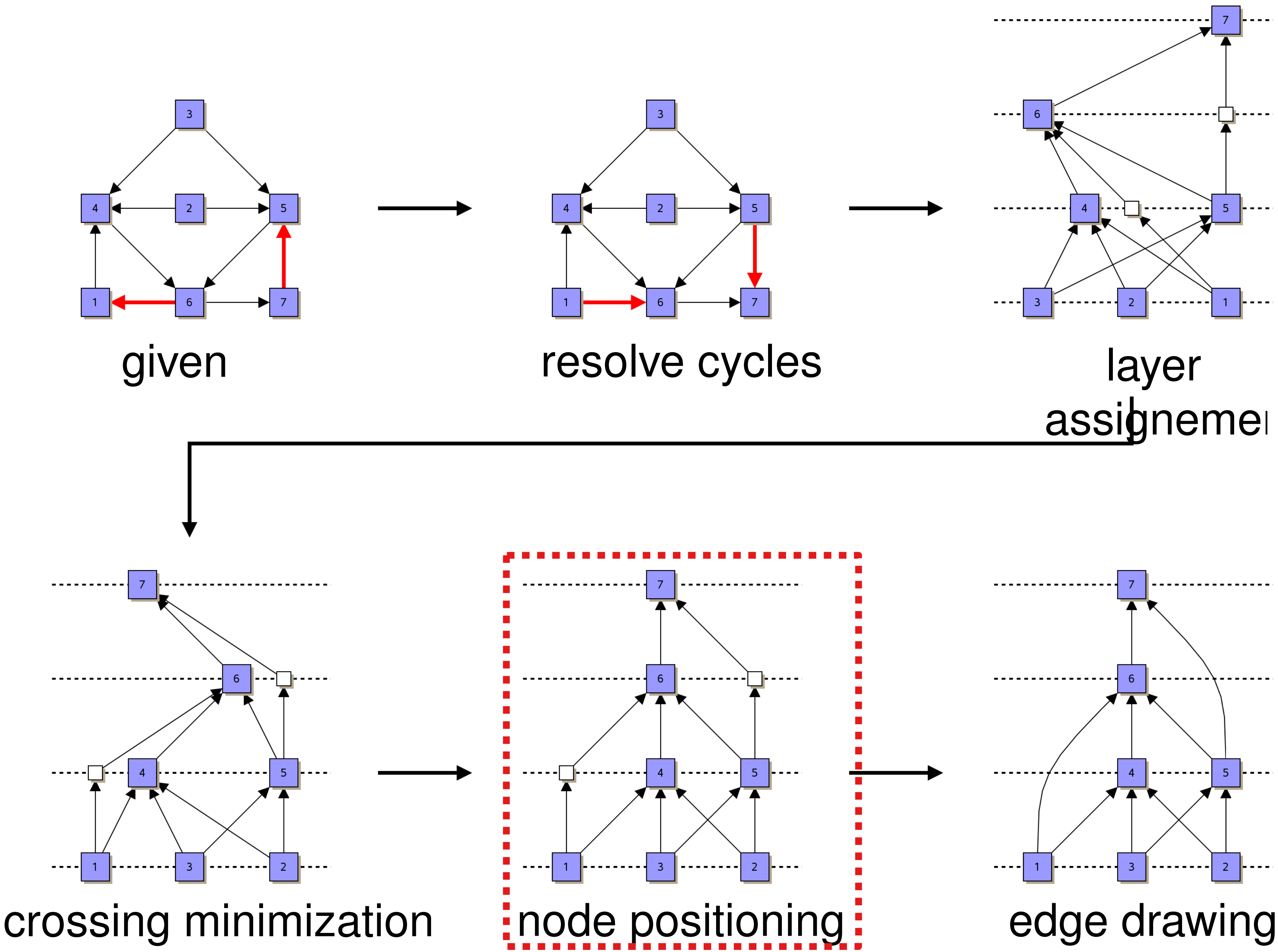
Idea: use quadratic Program

- let $p_{uv} = (u, d_1, \dots, d_k, v)$ path with k dummy nodes betwen u and v
- let $a_i = x(u) + \frac{i}{k+1}(x(v) - x(u))$ the x-coordinate of d_i when (u, v) is straight
- $g(p_{uv}) = \sum_{i=1}^{k}(x(d_i) - a_i)^2$
- minimize $\sum_{uv \in E} g(p_{uv})$
- constraints: $x(w) - x(z) \geq \delta$ for consecutive nodes on the same layer, w on the right of z (δ distance parameter)

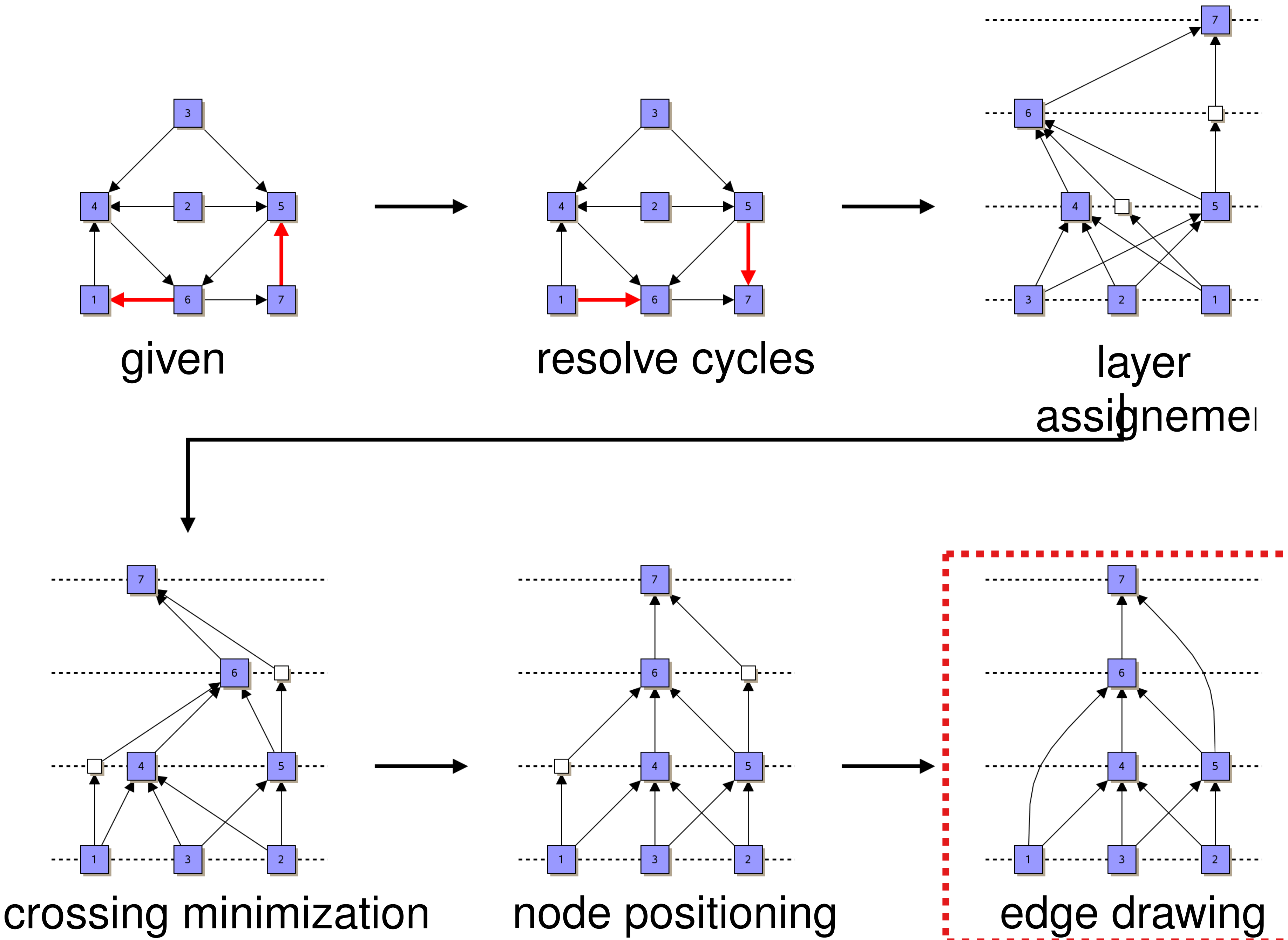
Properties:

- quadratic program is time-expensive
- straight edges increase width
- width can be exponential
- optimization function can be adapted to optimize ”verticality”

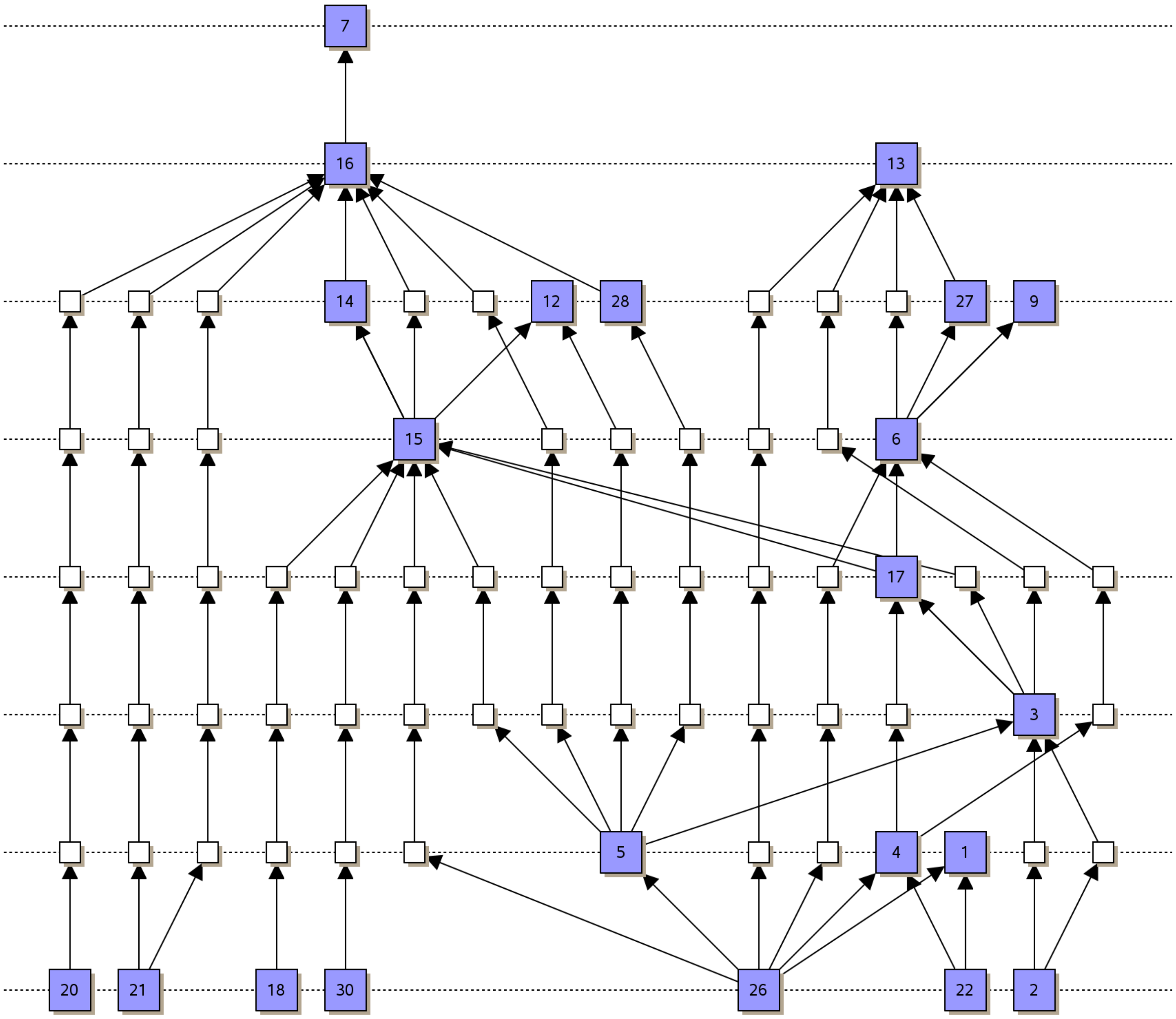
Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



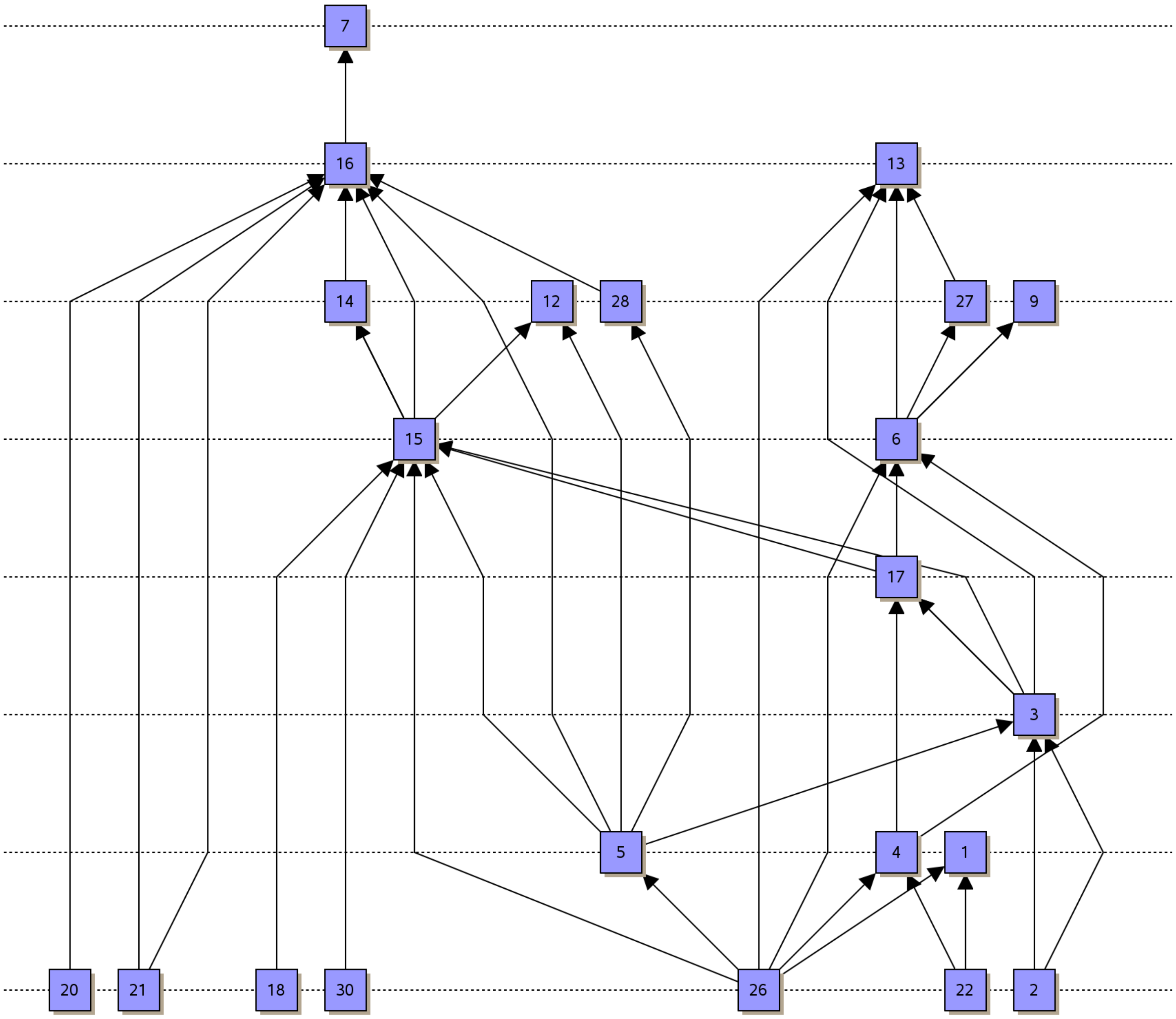
Sugiyama Framework (Sugiyama, Tagawa, Toda 1981)



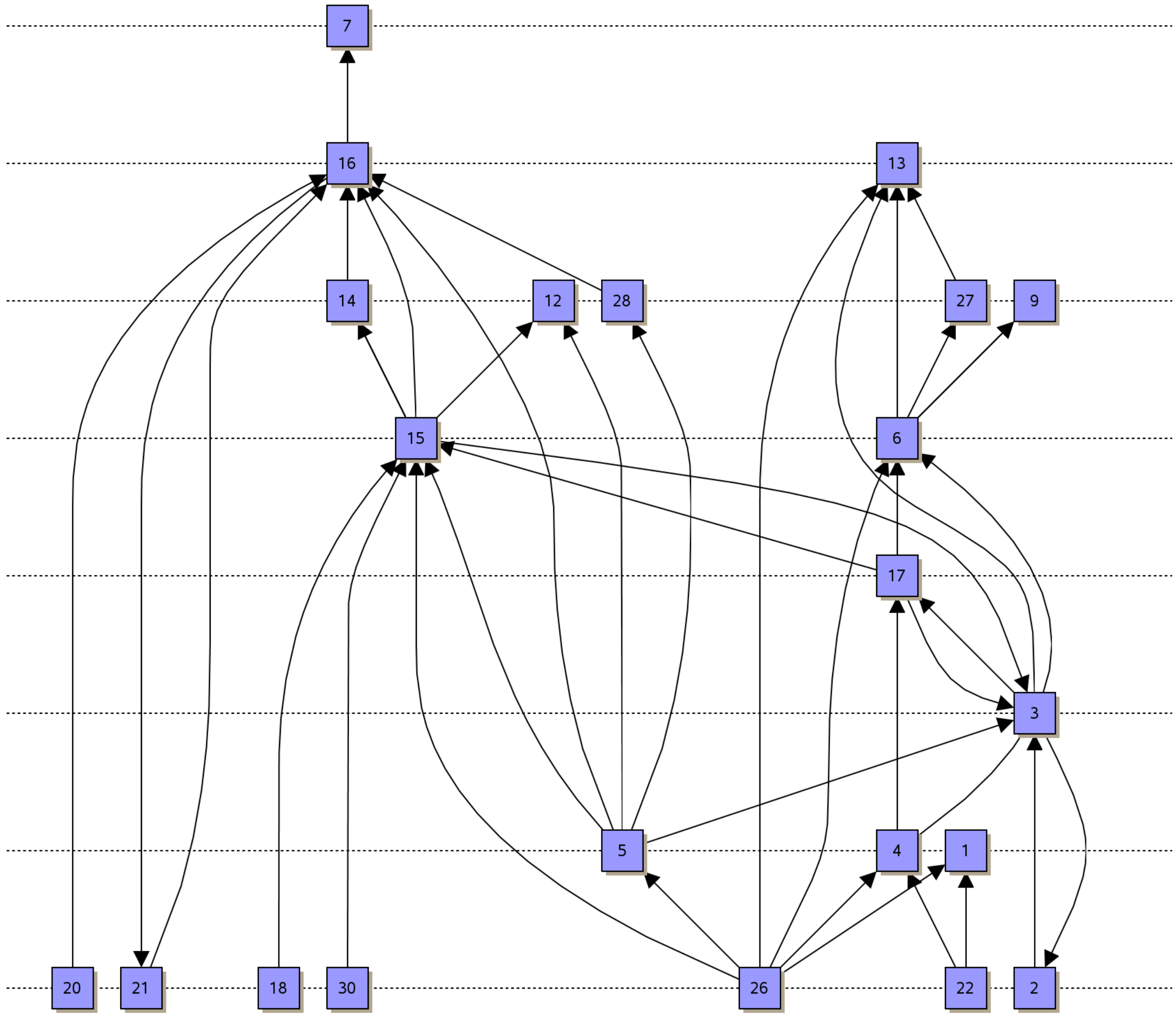
Edge Drawing



Edge Drawing

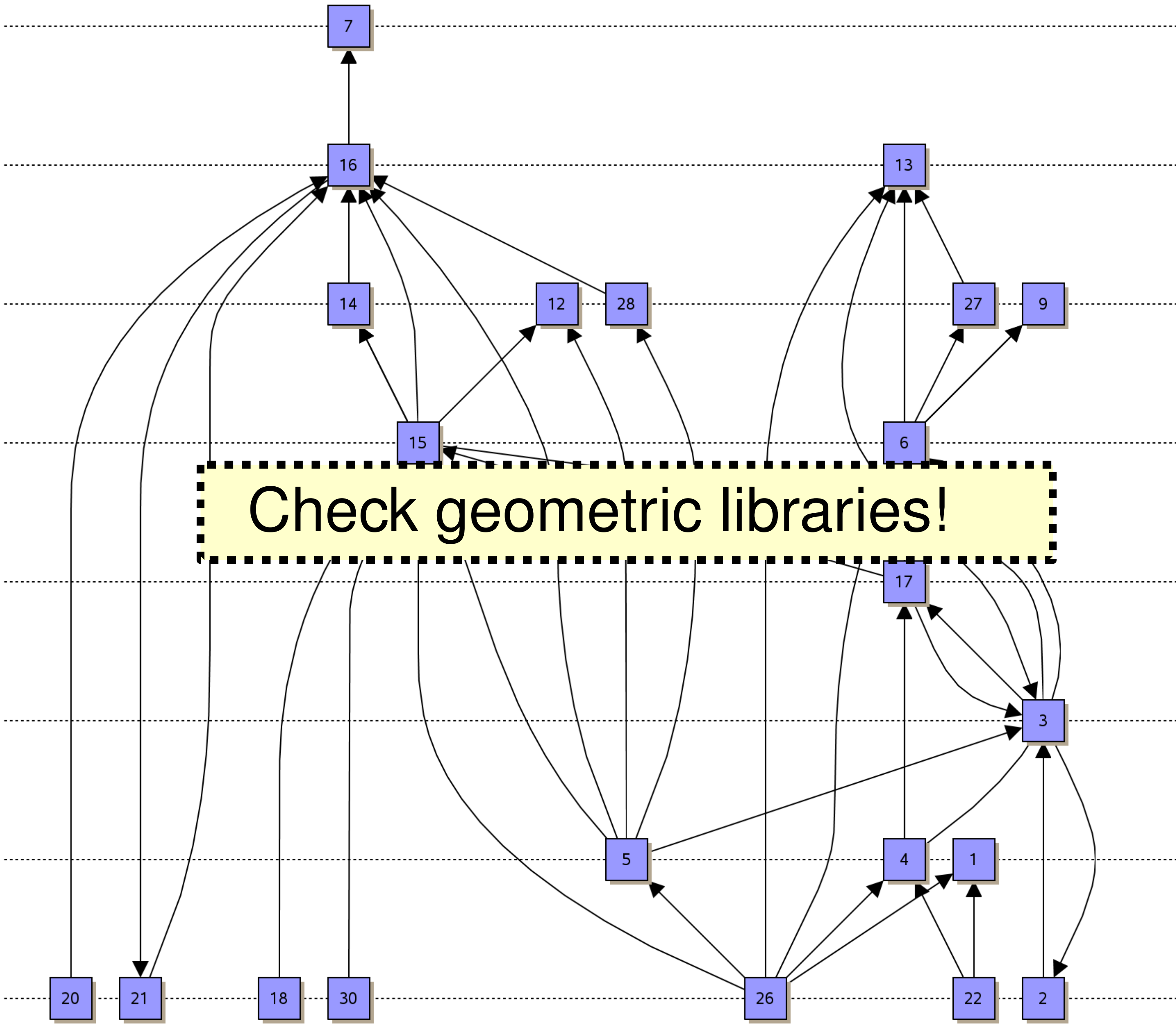


Edge Drawing



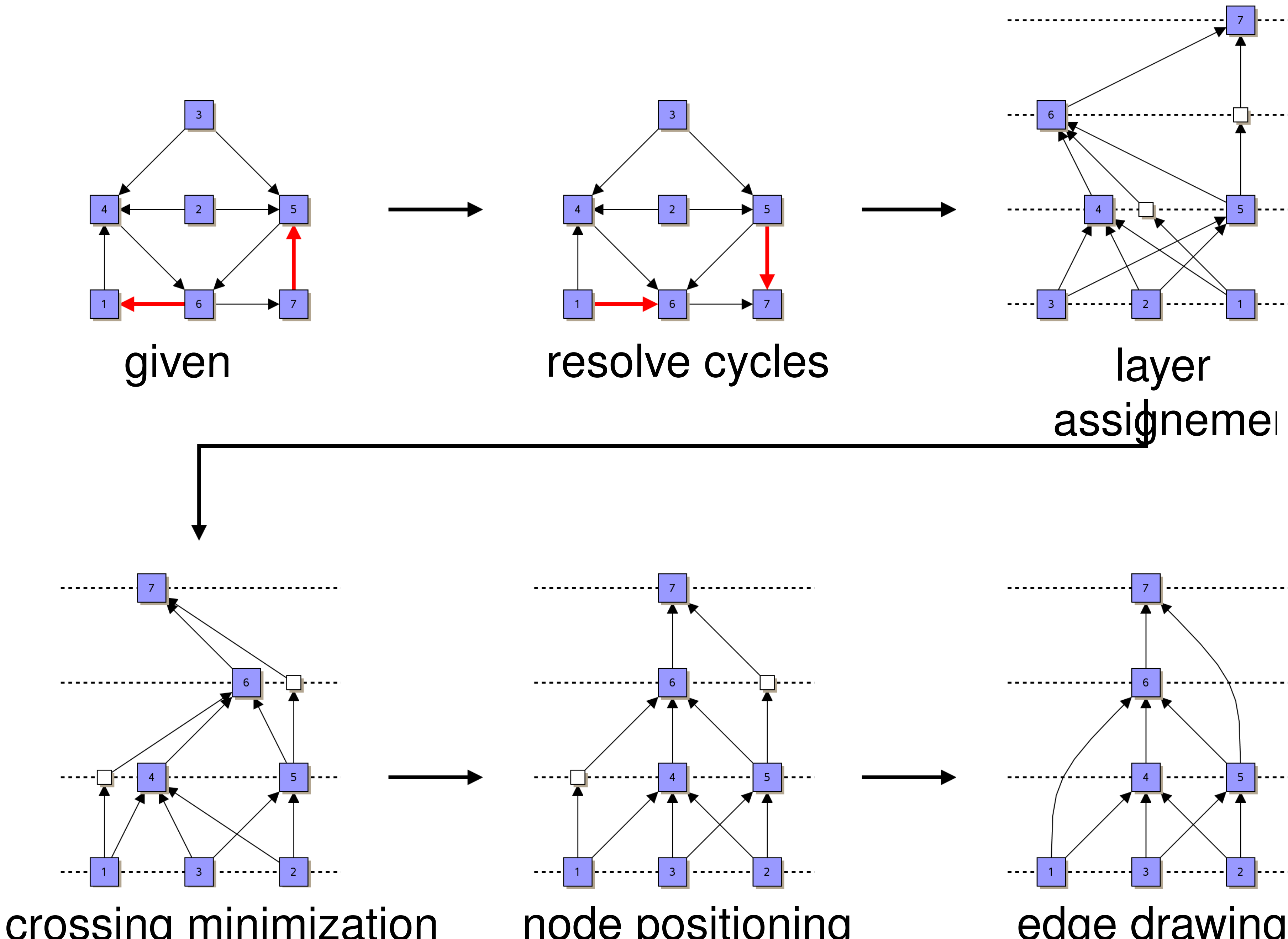
Possibility: Substitute polylines by Bézier curves

Edge Drawing

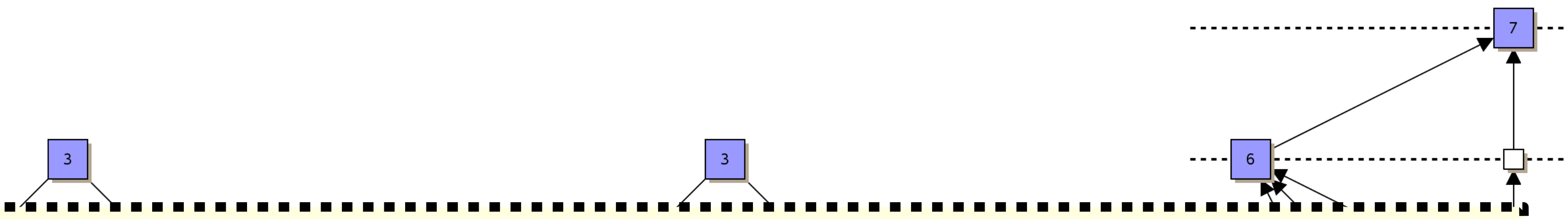


Possibility: Substitute polylines by Bézier curves

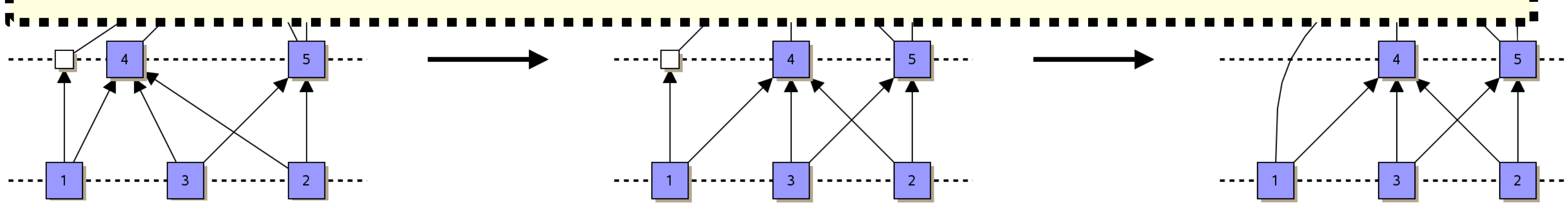
Summary


given
resolve cycles
layer assignement
crossing minimization
node positioning
edge drawing

Summary



- flexible **framework** to draw directed graphs
- sequential optimization of various criteria
- drawback: decisions taken on the previous steps influence the next steps and can not be undone
- modeling gives NP-hard problems, which can still can be solved quite well with heuristics
- many functionalities implemented in yEd and GraphVis



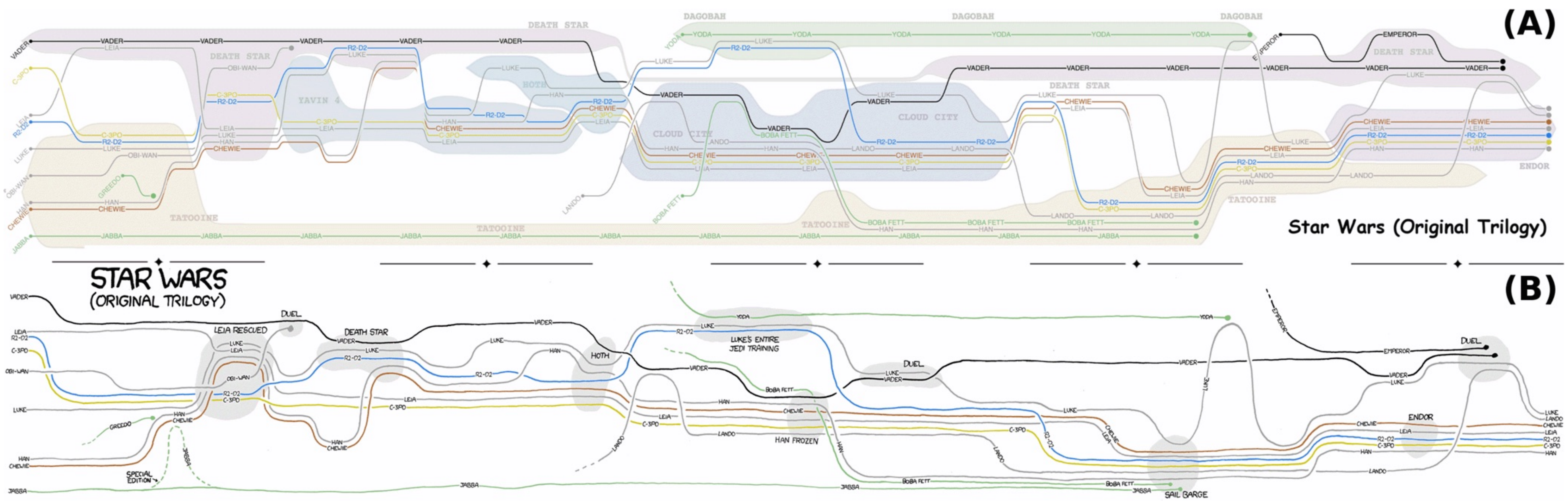
crossing minimization node positioning edge drawing

Reading

Additional Reading

Graph Drawing handbook: Chapter 13

Paper ”A framework and algorithms for circular drawings of graphs” J. Six, I. Tollis – fast computation of the number of crossings



Reading

Additional Reading

Graph Drawing handbook: Chapter 13

Paper ”A framework and algorithms for circular drawings of graphs” J. Six, I. Tollis – fast computation of the number of crossings

Next

Methods for visualization of multilayer/clustered networks