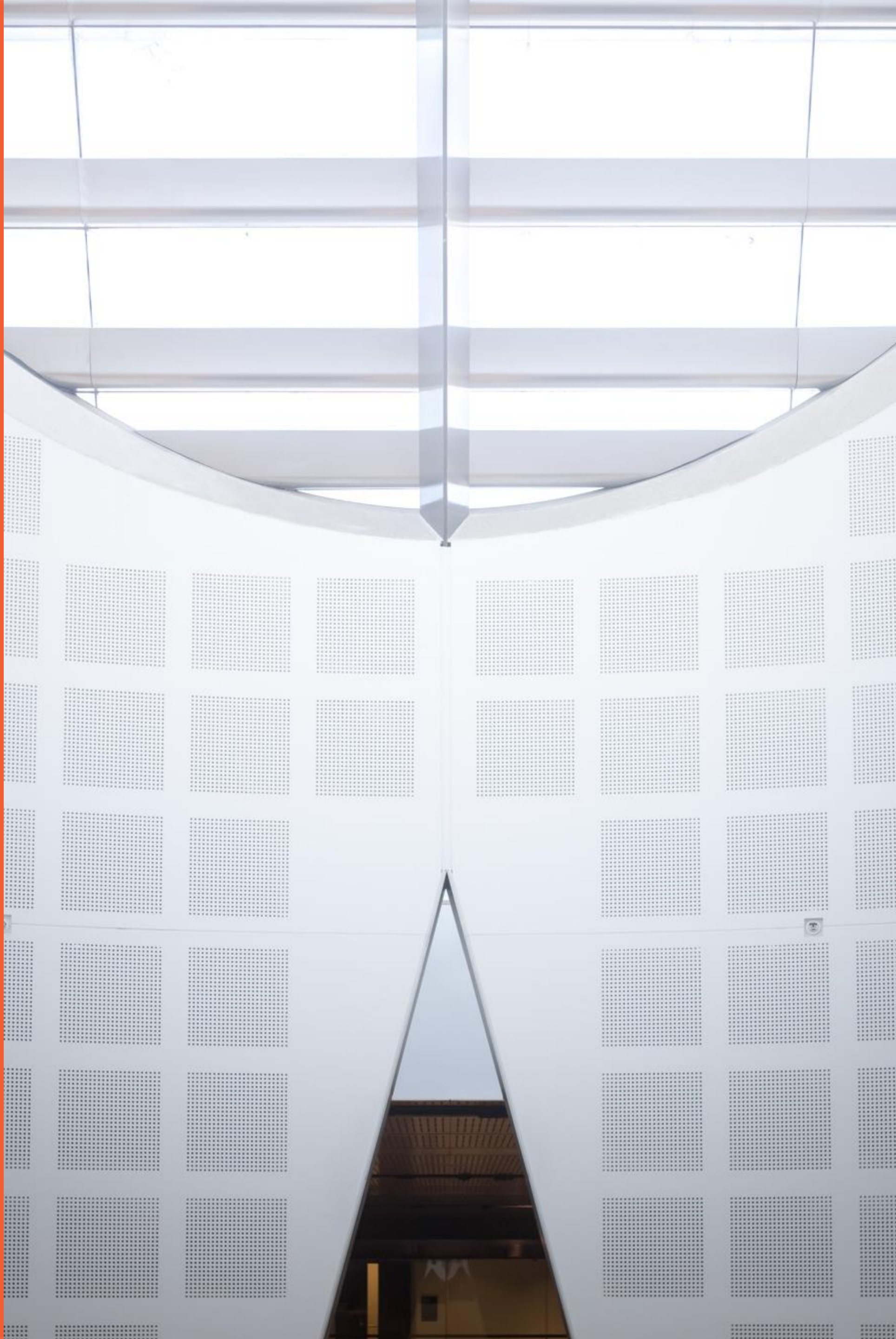
COMP5048
Visual Analytics

Week 4: Directed Graph Visualisation

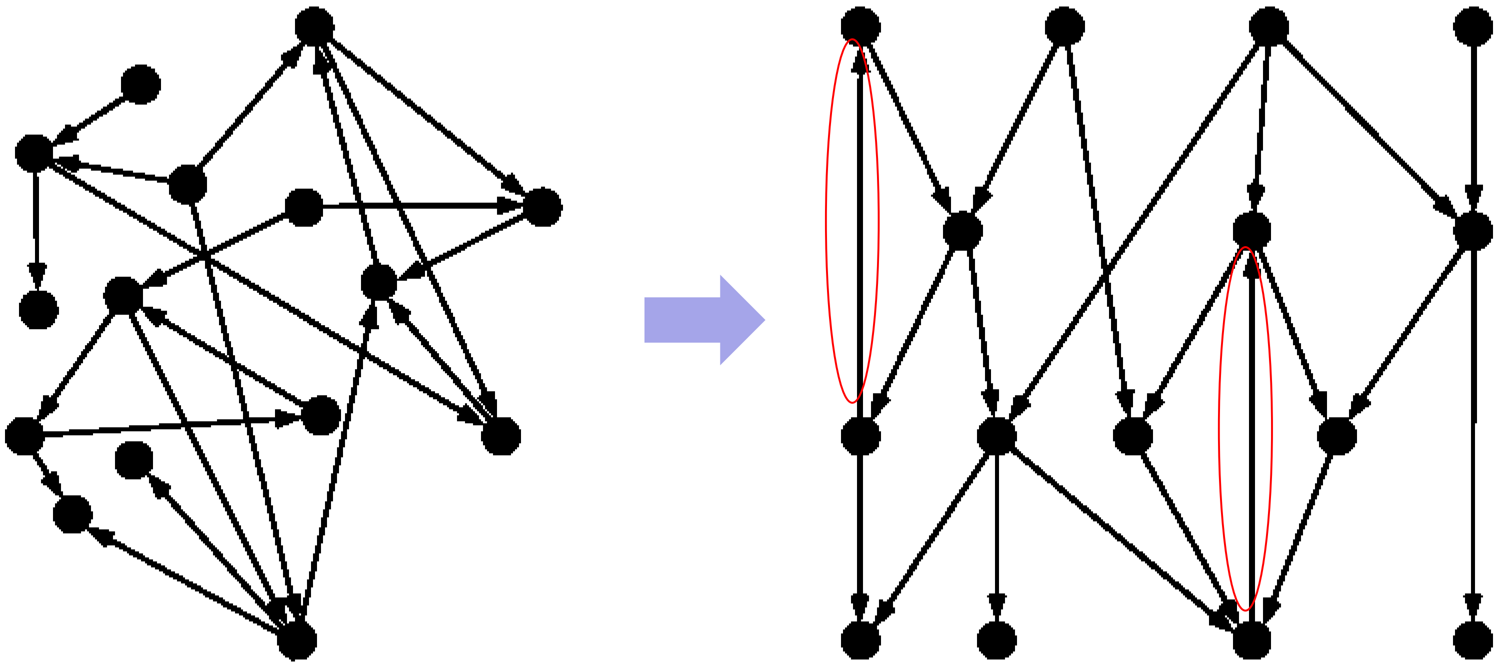
Professor Seokhee Hong
School of Information Technologies

THE UNIVERSITY OF SYDNEY



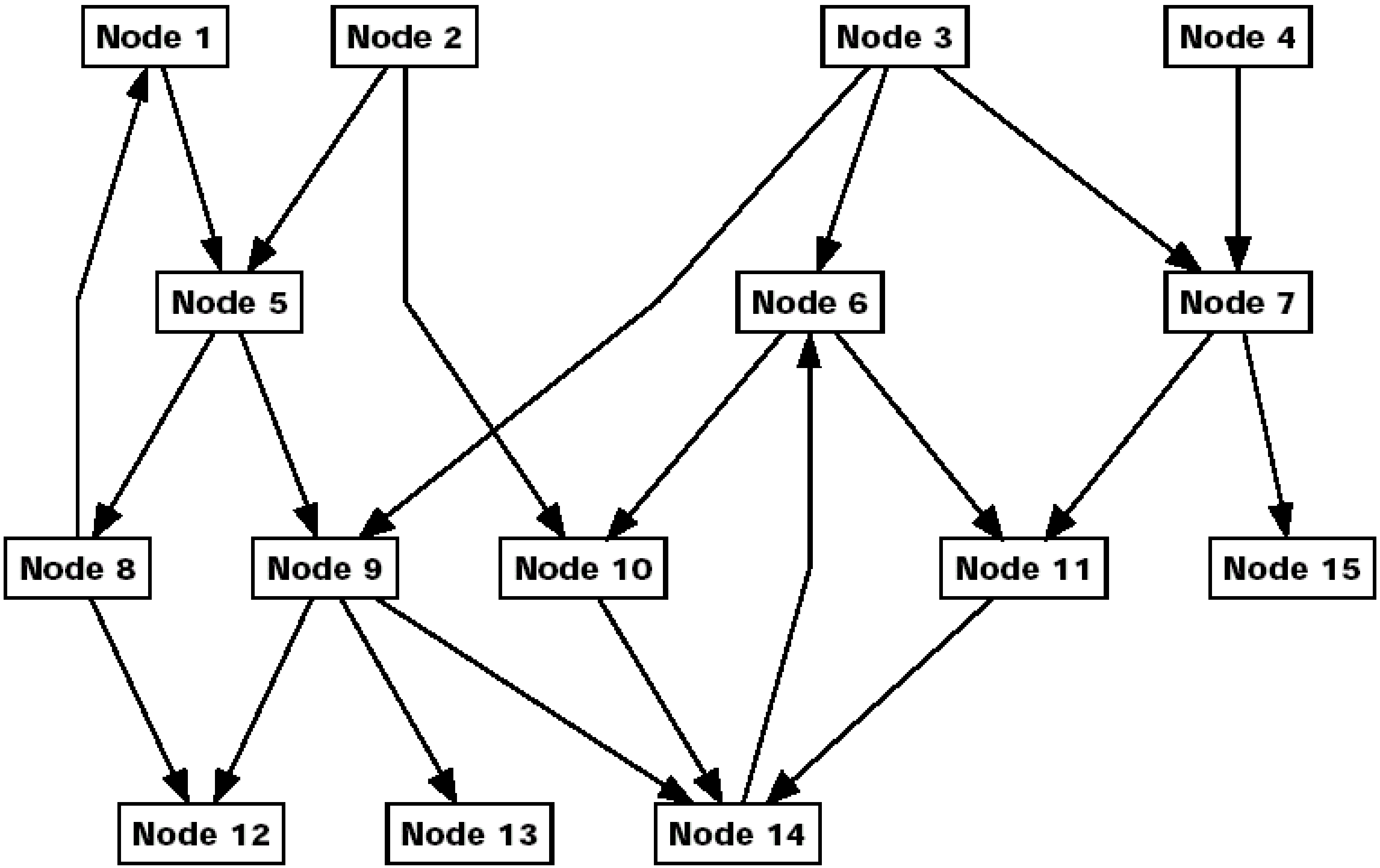
Copyright warning

Layered (Level/Hierarchical) Drawing



1. Edges pointing upward should be avoided.
2a. Nodes should be evenly distributed.
2b. Long edges should be avoided.
3. There should be as few edge crossings as possible.
4. Edges should be as straight/vertical as possible.

Layered Drawing



Sugiyama method

- Layered networks are often used to represent dependency relations.
- [Sugiyama *et al.* 1979]
  - few edge crossings
  - edges as straight as possible
  - nodes spread evenly over the page

- The Sugiyama method is useful for
  - dependency diagrams
  - flow diagrams
  - conceptual lattices
  - other directed graphs: acyclic or nearly acyclic.

Sugiyama Method

4 steps:

■step1. *Cycle removal*: make acyclic digraph

■step2. *Layer assignment*: assign y-coordinates

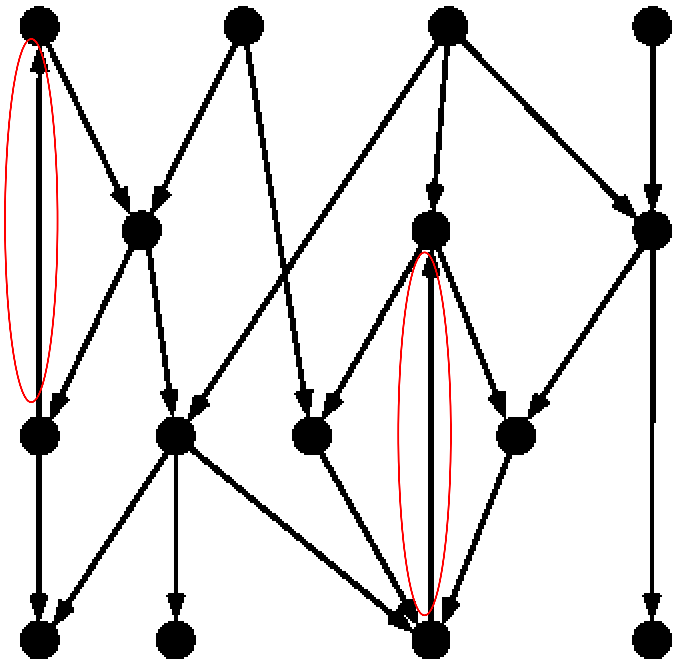
■step3. *Crossing reduction*: determine the order of vertices in each layer

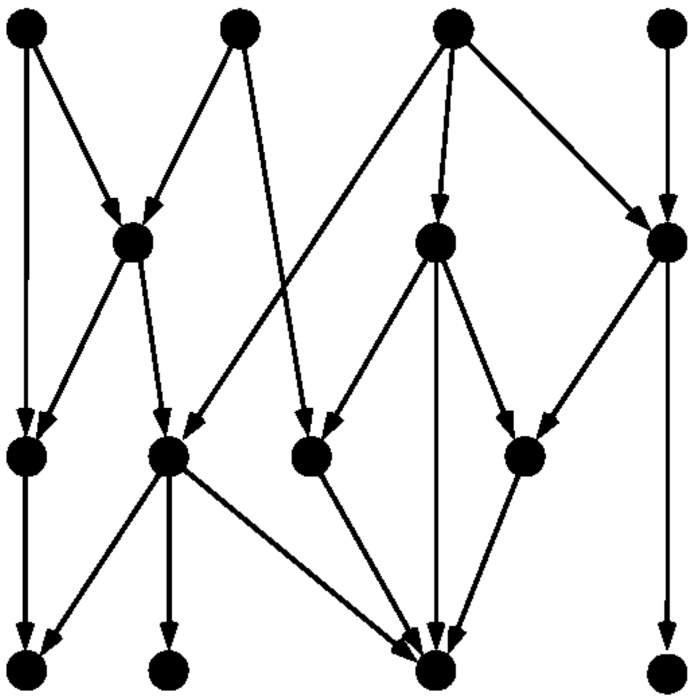
■step4. *Horizontal coordinate assignment*: assign x-coordinates (Straighten the long edges)

Step 1. Cycle Removal

Step 1. Cycle Removal

- Input graph may contain cycles
  1. make an acyclic digraph by <u>*reversing*</u> some edges
  2. draw the <u>acyclic graphs</u>
  3. render the drawing with the reversed edges





Acyclic graph by reversing two edges

Step 1. Cycle Removal

- Each cycle must have at least one edge against the flow
  - need to keep the number of edges against the flow *small*
- main problem: how to choose the *small set of edges R*
  - Feedback set: set of edges R whose *reversing* makes the digraph acyclic
  - Feedback arc set: set of edges whose *removal* makes the digraph acyclic

- Maximum acyclic subgraph problem
  - find a maximum set *Ea* such that the graph(V, Ea) contains no cycles : NP-hard [Karp72, GJ91]
- Feedback arc set problem
  - find a minimum set *Ef* such that the graph(V, E\Ef) contains no cycles : NP-complete [GJ79]

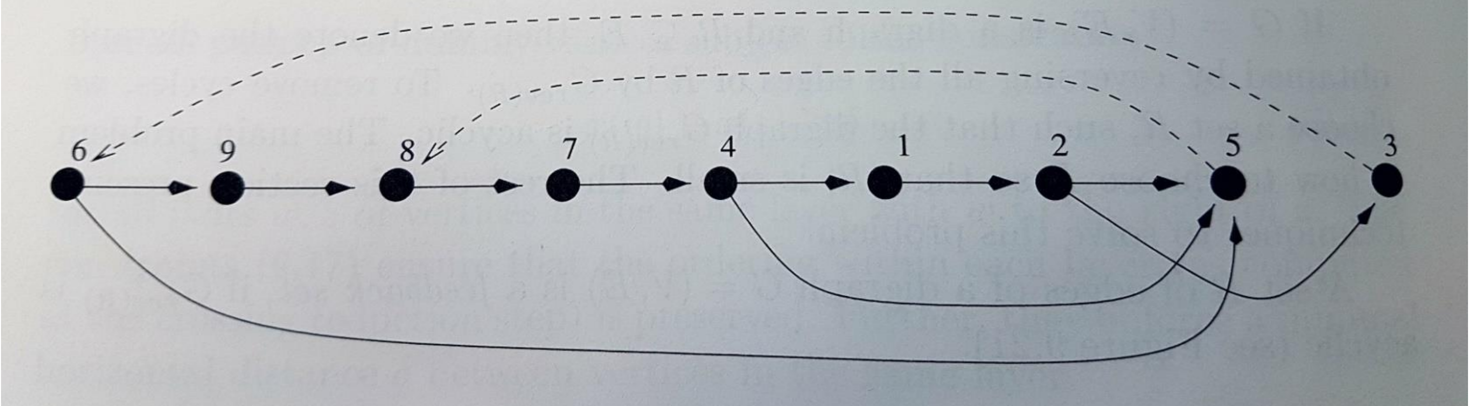


Figure 9.22: Leftward edges shown with dashed lines.

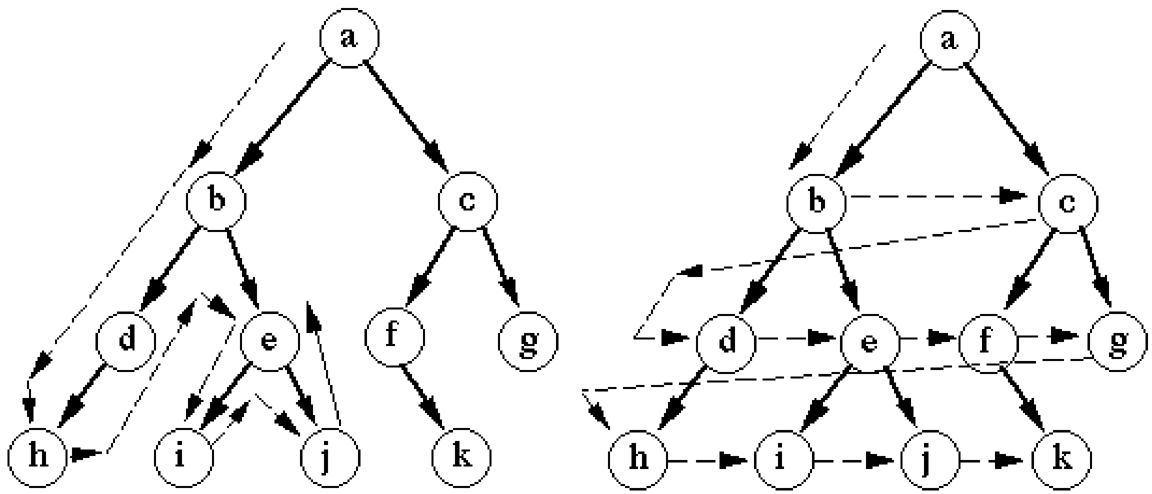
- Feedback set problem:
  - equivalent to finding a vertex sequence with *as few leftward edges as possible*
  - *S=(v1, v2, …, vn): a* vertex sequence of a digraph G
  - set of leftward edges *((vi, vj) with i > j*) for a vertex sequence forms a feedback set

  - i.e, a linear ordering problem:
  - find a linear ordering ***o*** of the vertices, such that the # of edges *(u,v),* ***o****(u) >* ***o****(v) is minimized*.

1. Simple Heuristics

1. Depth First Search (or Breadth First Search)

- Compute an ordering ***o*** of vertices using DFS/BFS.
- then delete edges *(u,v)* with ***o****(u)* > ***o****(v)*
- poor performance: reverse *|E|-|V|-1 edges* in worst case
- runs in linear time



Depth-first search

Breadth-first search

1. Simple Heuristics

2. Simpler Heuristic [Berger, Shor 90]

guarantees acyclic set *Ea* of size at least ½|*E*|

- Delete either incoming or outgoing edges for each vertex
- Linear time

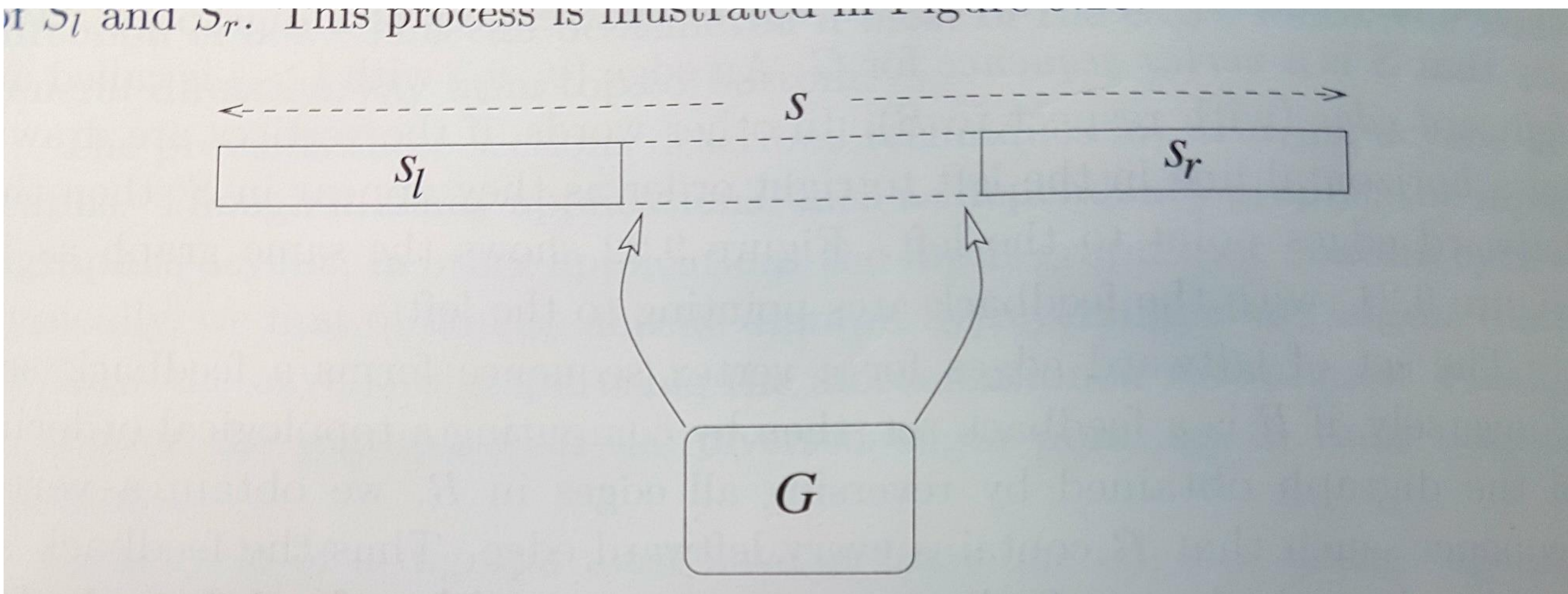
Algorithm 8: A Greedy Algorithm

$E_a = \emptyset$;
foreach $v \in V$ **do**
  if $|\delta^+(v)| \geq |\delta^-(v)|$ **then**
    append $\delta^+(v)$ to E_a;
  else
    append $\delta^-(v)$ to E_a;
  delete $\delta(v)$ from G;

- Set of outgoing edges from v
- Set of incoming edges into v

2. Greedy Heuristic [Eades et al. 93]

- Source & sink play a special role:
  – source: vertex without incoming edges
  – sink: vertex without outgoing edges
  – edges incident to source & sink cannot be part of a cycle
- Successively remove vertices from G
- Add each in turn, to one of two lists *Sl & Sr*:
  – either the end of *Sl* or the beginning of *Sr*



2. Greedy Heuristic [Eades et al. 93]

- All sources should be added to *Sl*
- All sinks should be added to *Sr*
- Choose a vertex *v* whose *outdeg(v)-indeg(v)* is maximized and add to *Sl*

- Can be implemented in linear time

- Sparse graph: Ea with at least 2/3|E| (max. degree <=3)
- Performance:

$$|E_a| \geq \frac{|E|}{2} + \frac{|V|}{6}.$$

Algorithm 9.4 *Greedy-Cycle-Removal*
Input: digraph G
Output: vertex sequence S for G

1. Initialize both S_l and S_r to be empty lists.
2. **while** G is not empty **do**
   (a) **while** G contains a sink **do**
      Choose a sink u, remove it from G, and prepend it to S_r. (Note: isolated vertices are removed from G and prepended to S_r at this stage.)
   (b) **while** G contains a source **do**
      Choose a source v, remove it from G, and append it to S_l.
   (c) **if** G is not empty **then**
      Choose a vertex u, such that the difference $outdeg(u) - indeg(u)$ is maximum, remove it from G, and append it to S_l.
3. Concatenate S_l with S_r to form S. □

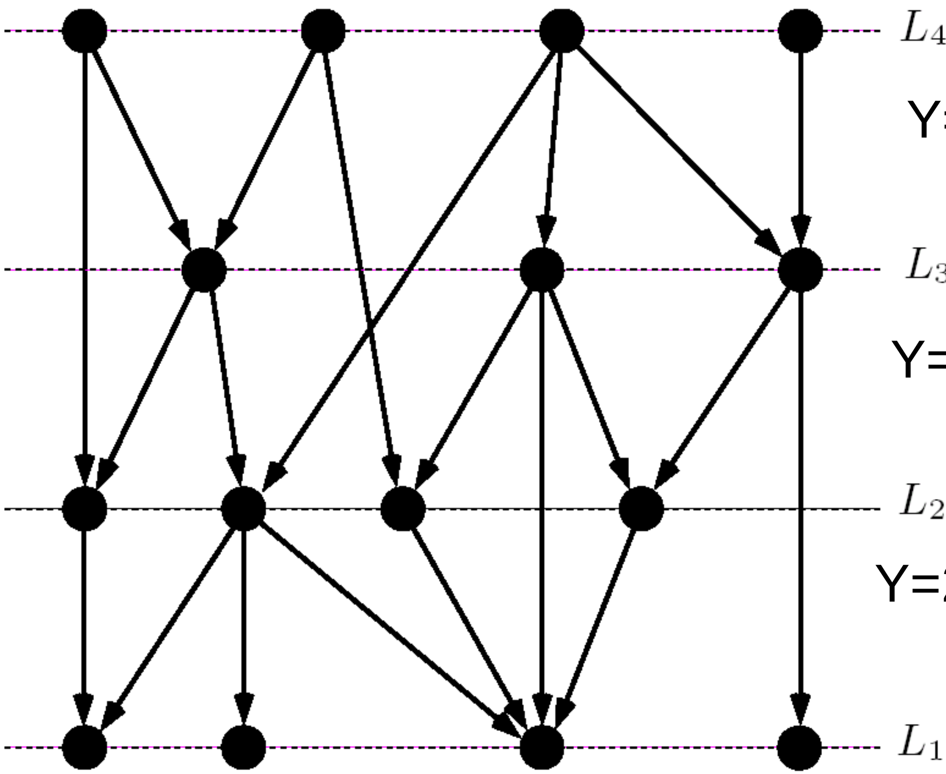
Step 2. Layer Assignment

Step 2. Layer Assignment

- Layering: *partition V into L1, L2, …, Lh*
- Layered (di)graph: digraph with layers

- Height *h*: # of layers
- Width *w*: # of vertices with largest layer
- Span of an edge *(u,v): h(u)-h(v)*
- Proper digraph: no edge has a span > 1

- Some application, vertices are preassigned to layers



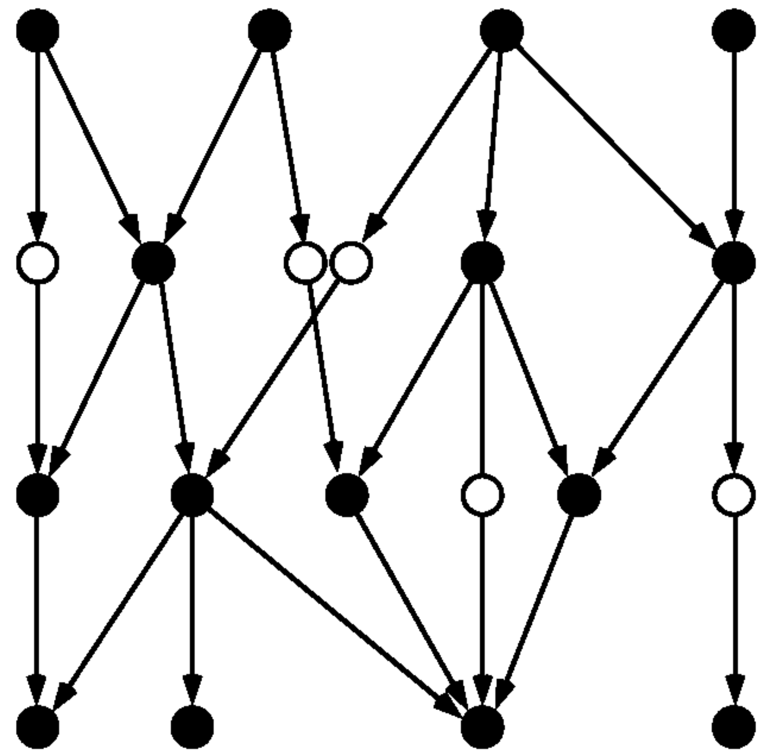
Y=4

Y=3

Y=2

Y=1

Layering



Introducing ***dummy*** vertices

Step 2. Layer Assignment

- Requirements
  1. Layered digraph should be compact: height & width
  2. The layering should be proper: add dummy vertices
  3. The number of dummy vertices should be small
     A. time depends on the total number of vertices
     B. bends in the final drawing occur only at dummy vertices

- Methods
  1. Longest path layering: minimize height
  2. Layering to minimize width
  3. Minimize the number of dummy vertices

1. Longest path layering

- Minimize the height

- Place all sinks in layer L_1
- Each remaining vertex v is placed in layer L_{p+1}, where the longest path from v to a sink has length p

$$y(u) := \max\{i \mid v \in N^+(u) \text{ and } y(v) = i\} + 1$$

$$N^+(u) := \{v \in V \mid \exists (u, v) \in E\}$$

- linear time using topological ordering [Melhorn 84]
- Main drawback: too wide

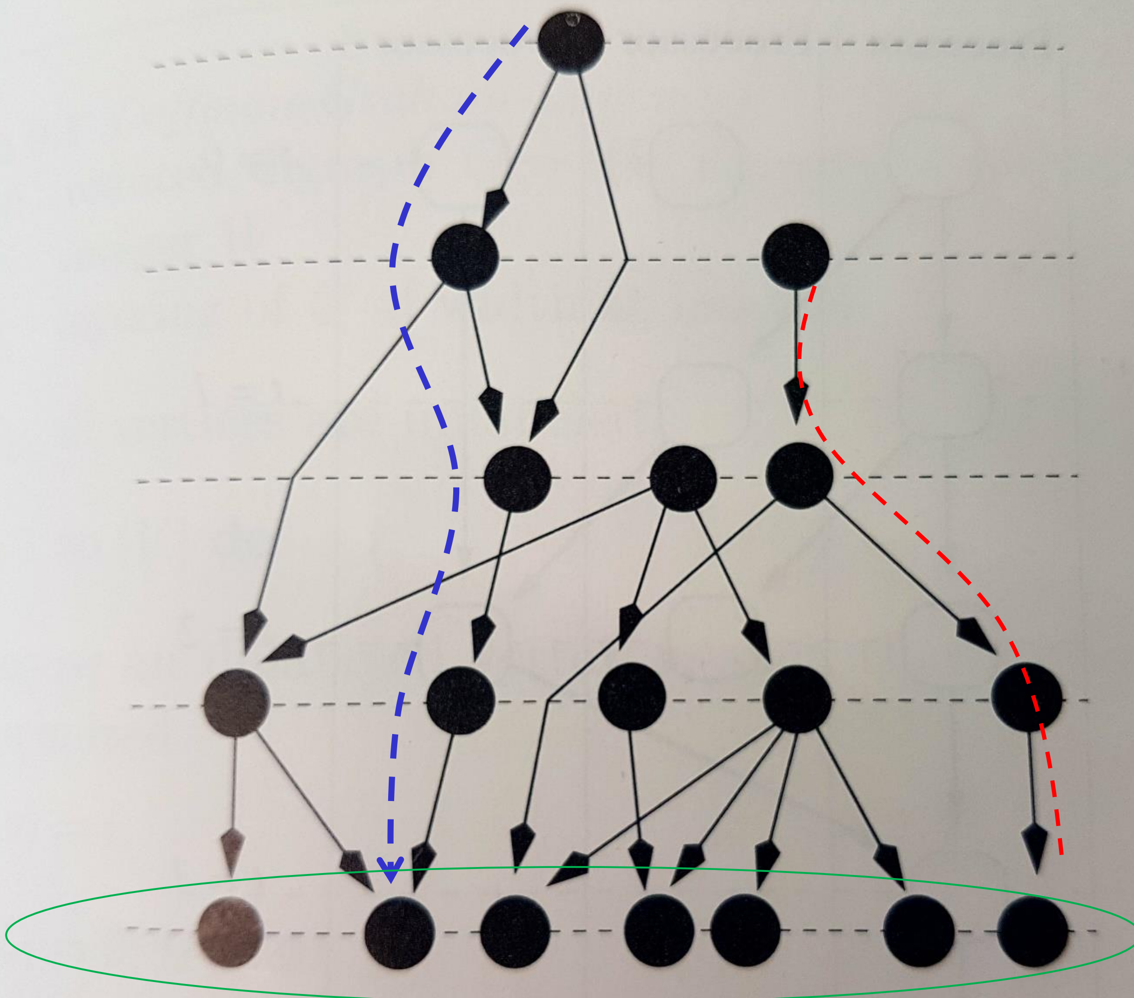


Figure 9.9: A longest path layering.

2. Layering to minimize width

- Finding a layering with minimum *width W* subject to minimum *height H*: *Precedence-constrained multiprocessor scheduling problem*
  - *assign each task to one of W processors, so that all tasks are completed in time H:* NP-complete [Karp 72, Garey and Johnson 79]

- Coffman-Graham Layering [Coffman Graham 72]
  - Input: reduced graph G (no transitive edges) and W
  - Output: layering of G with width at most W
  - Aim: try to ensure the height of the layering is kept small
  - Two phases

    1. Order the vertices: based on the distance from the source
    2. Assign layers: vertices with large distances from the sources will be assigned to layers as close to the bottom as possible

- Width: does not count dummy vertices

Coffman-Graham Layering

- First phase: *lexicographical ordering*

$S \prec T$ if either

1. $S = \emptyset$ and $T \neq \emptyset$, or
2. $S \neq \emptyset$, $T \neq \emptyset$ and $\max(S) < \max(T)$, or
3. $S \neq \emptyset$, $T \neq \emptyset$, $\max(S) = \max(T)$ and $S \setminus \{\max(S)\} \prec T \setminus \{\max(T)\}$

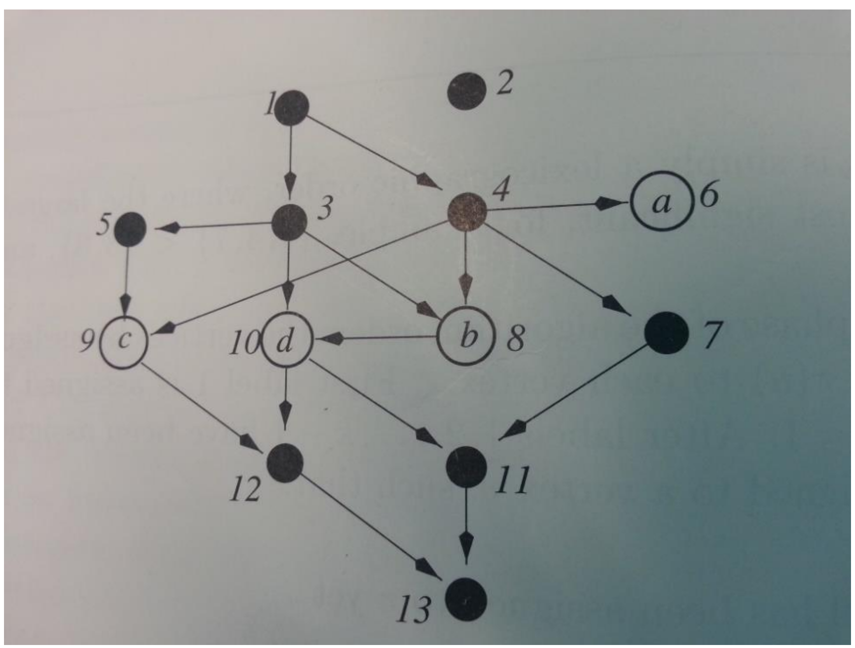
  – $\{1,4,\underline{7}\} < \{3,\underline{8}\}$

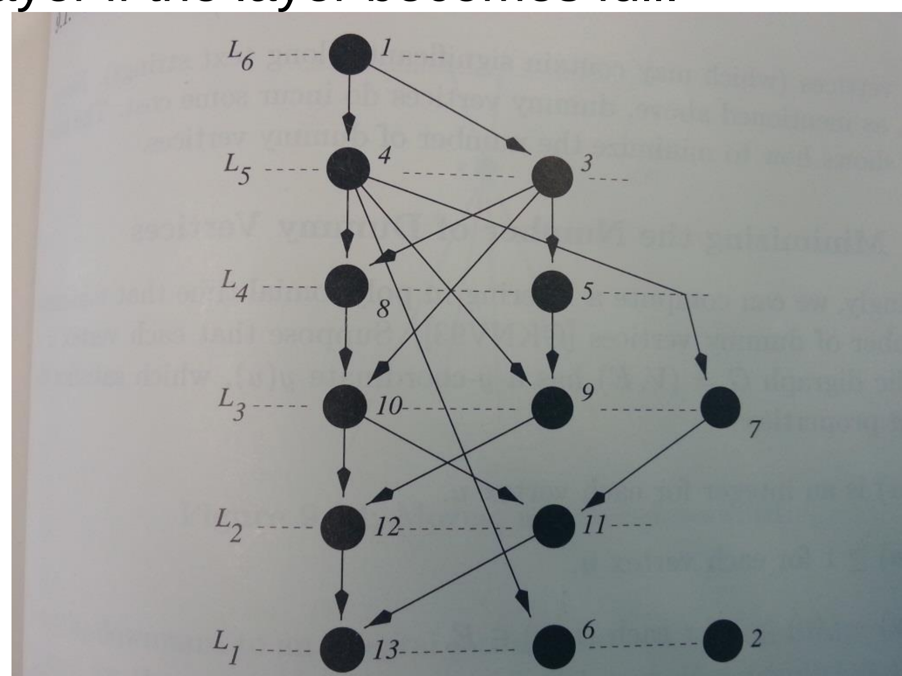
  – $\{3,\underline{4},9\} < \{1,\underline{5},9\}$

- Second phase: ensure that no layer receive more than W vertices
- [Lam, Sethi 77] height: not too large

$$h \leq \left(2 - \frac{2}{w}\right) h_{opt}$$

- Step 1: labelling vertices
  - source: label 1
  - choose a vertex v: form a tuple with labels of all incoming neighbours
  - assign labels to vertices based on lexicographical ordering (choose min.)
  - (e.g.) a<b: {4} < {3,4},   c<d: {4,5} < {3,8}
- Step 2: assign vertices to each layer
  - start from the bottom layer (sink)
  - choose a vertex v: all outgoing neighbours have been placed.
  - if more than one such vertex, choose the one with the largest label.
  - check width w: move to the next layer if the layer becomes full.





Coffman-Graham Layering

Algorithm 9.1 *Coffman-Graham-Layering*

Input: reduced digraph $G = (V, E)$, and a positive integer W

Output: layering of G of width at most W

1. Initially, all vertices are unlabeled.
2. **for** $i = 1$ **to** $|V|$ **do**
   (a) Choose an unlabeled vertex v, such that $\{\pi(v) : (u, v) \in E\}$ is minimized
   (b) $\pi(v) = i$.
3. $k = 1$; $L_1 = \emptyset$; $U = \emptyset$.
4. **while** $U \neq V$ **do**
   (a) Choose $u \in V - U$, such that every vertex in $\{v : (u, v) \in E\}$ is in U, and $\pi(u)$ is maximized
   (b) **if** $|L_k| < W$ **and** for every edge (u, w), $w \in L_1 \cup L_2 \cup \ldots L_{k-1}$
   then add u to L_k
   else $k = k + 1$, $L_k = \{u\}$
   (c) Add u to U. □

3. Minimizing # of dummy vertices

- one can compute a layering in polynomial time that minimizes the number of dummy vertices [Gansner et al.93]
- $$f = \sum_{(u,v) \in V} (y(u) - y(v) - 1)$$
- f: sum of vertical spans of the edges in the layering

- # of edges : (# of dummy vertices)

- Layer assignment problem is reduced to choosing y-coordinates to minimize f
- <u>Integer linear programming </u>problem

Step 3. Crossing Minimisation

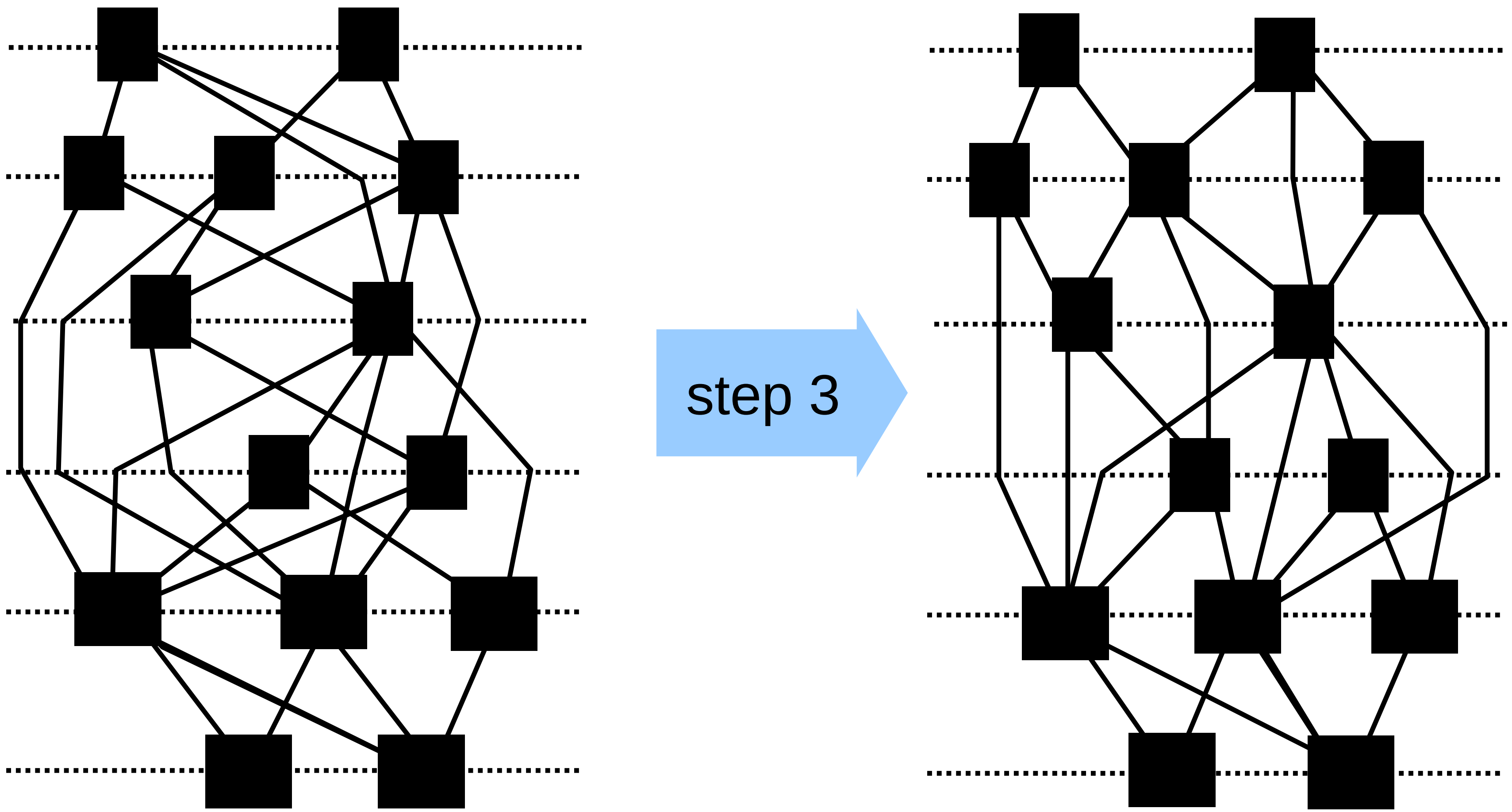
Step 3. Crossing Minimization

- Input: proper layered graph
- # of edge crossings does not depend on the precise position of the vertices, but only the ordering of the vertices within each layer (i.e., combinatorial, not geometric)

- NP-complete, even only two layers [Garey,Johnson 83]:
- 2-layer crossing minimization problem
- Heuristics: Layer-by-layer sweep

- **One-sided 2-layer crossing minimization**
  – 1. Sorting methods: heuristics
  – 2. Barycenter method: approximation algorithm
  – 3. Median method: approximation algorithm
  – 4. Integer programming method: exact algorithm

Crossing Minimization: ordering



21 edge crossings

5 edge crossings

Layer-by-layer sweep

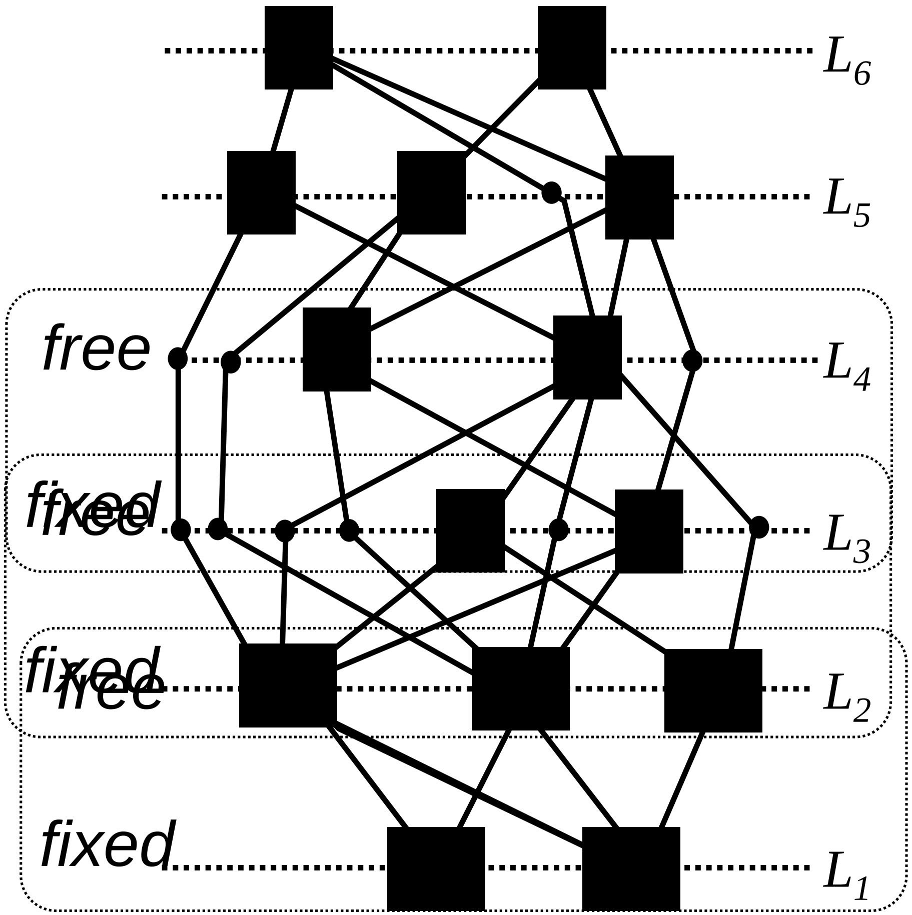
- A vertex ordering of layer L_1 is chosen
- *For i = 2, 3, …, h*
  - The vertex ordering of L_{i-1} is ***fixed***
  - Reordering the vertices in layer L_i to reduce edges crossings between L_{i-1} and L_i

- One-sided Two layer crossing minimization problem:

Given a fixed ordering of L_{i-1},

choose a vertex ordering of Layer L_i to minimize # of crossings

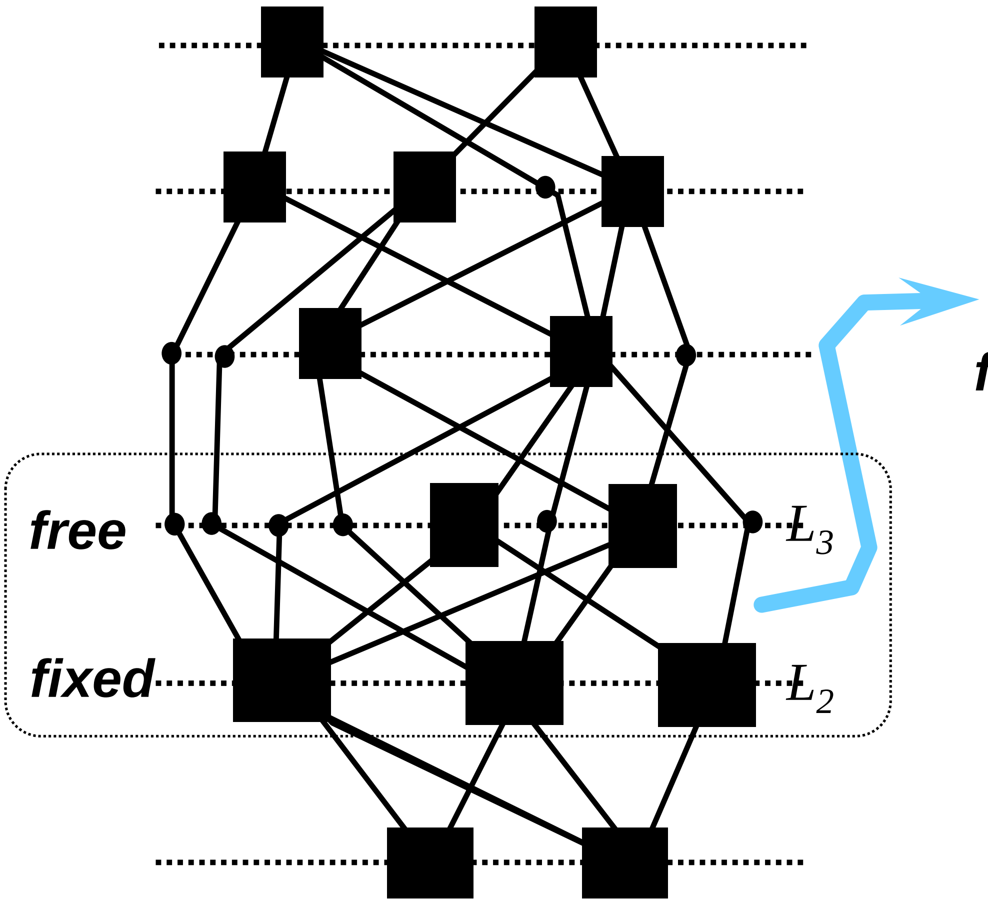
Layer-by-layer sweep



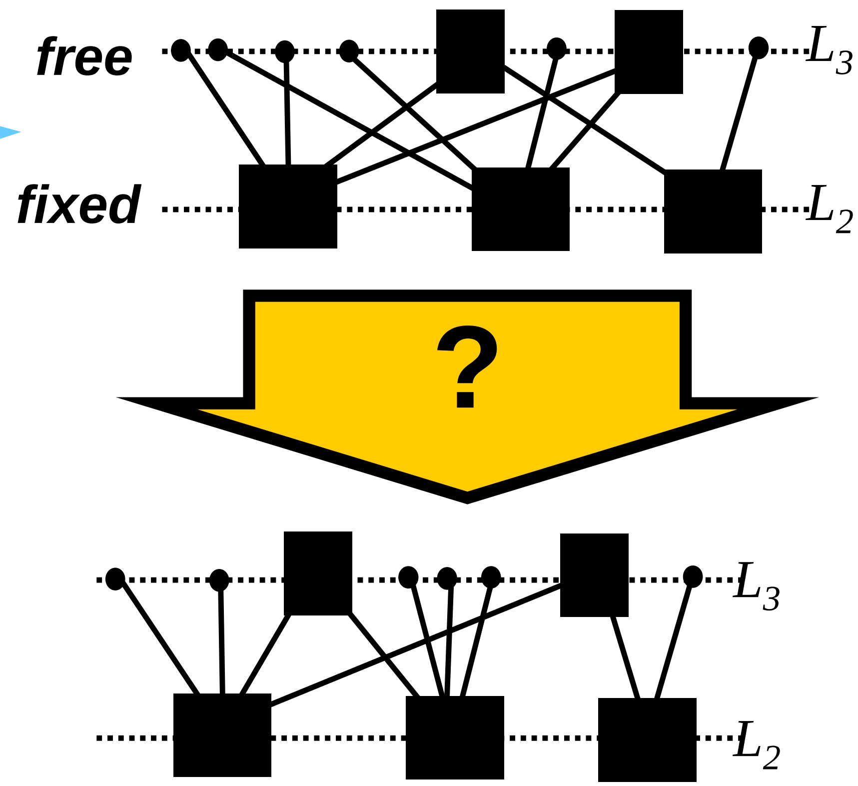
- "layer-by-layer sweep"
- from bottom to top

- At each stage of the sweep, we:
  - hold one layer fixed, and
  - Re-arrange the nodes in the layer above to avoid edge crossings.

Two layer crossing minimisation problem

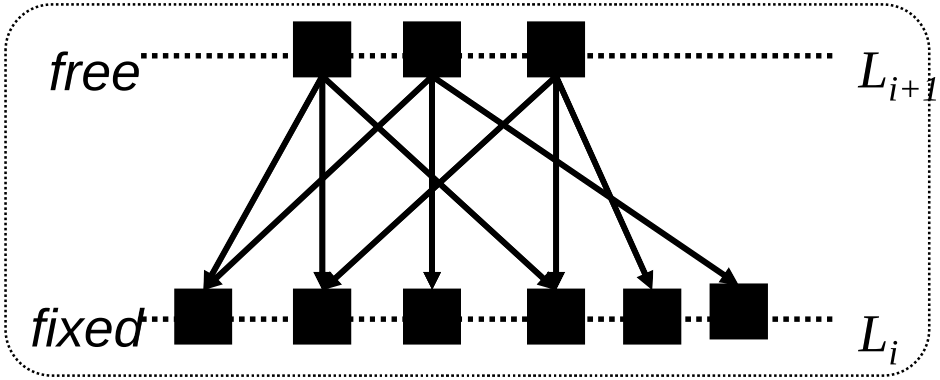


The difficult part is to re-arrange the free layer



One-sided two-layer crossing minimization

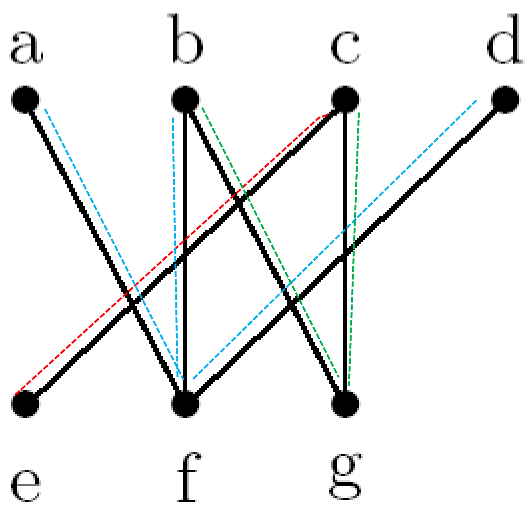
- The problem of finding an optimal solution is *NP-hard*.



- *Approximation algorithms*
  1. *Barycenter method*: place each free node at the barycenter of its neighbours.
  2. *Median method*: place each free node at the median of its neighbours.

Crossing Number

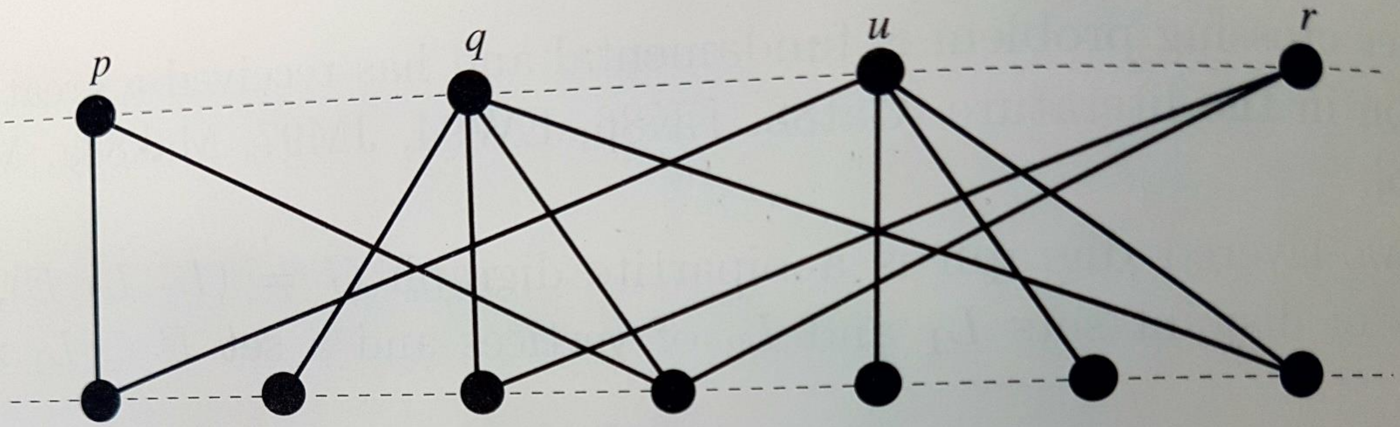
- **Crossing number c_{uv}**
  - – # of crossings that edges incident to u make with edges incident v, when $x2(u) < x2(v)$
  - – # of pairs (u,w), (v,z) of edges with $x1(z) < x1(w)$



C	e	f	g
e	0	2	1
f	1	0	2
g	0	3	0

***C**ef=2, **C**fg=2*

Crossing Number



(a)

	p	q	u	r
p	0	2	1	1
q	5	0	6	3
u	6	9	0	6
r	2	3	2	0

(b)

Bubble-sort

- Key idea: To sort a sequence of n comparable elements
  - Scan the sequence n-1 times
  - In each step of a scan, compare the current element with the next and swap them if they are out of order
    - by the end of the scan, the largest element reached the last position
    - the next scan works on a slightly shorter part

- Slow: $O(n^2)$ runtime

Example Bubble-sort

- First Pass:

(5 1 4 2 8) → (1 5 4 2 8)

(1 5 4 2 8) → (1 4 5 2 8)

(1 4 5 2 8) → (1 4 2 5 8)

(1 4 2 5 8) → (1 4 2 5 8)

- Second Pass:

(1 4 2 5 8) → (1 4 2 5 8)

(1 4 2 5 8) → (1 2 4 5 8)

(1 2 4 5 8) → (1 2 4 5 8)

- Third Pass:

(1 2 4 5 8) → (1 2 4 5 8)

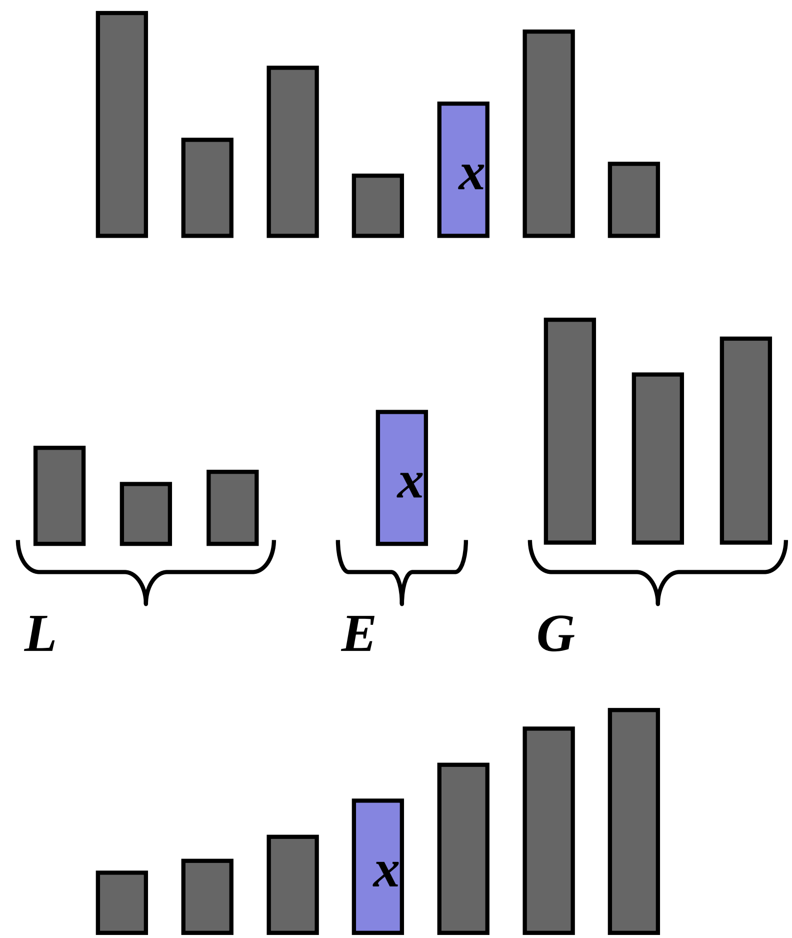
(1 2 4 5 8) → (1 2 4 5 8)

- Fourth Pass:

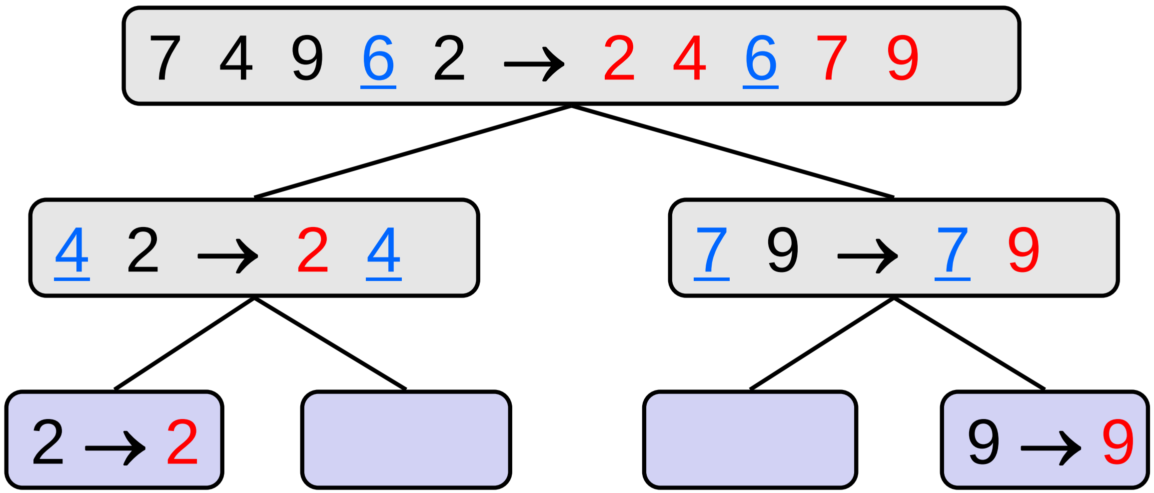
(1 2 4 5 8) → (1 2 4 5 8)

Quick-Sort

- divide-and-conquer algorithm
- Divide:
  - pick a *random* element x (<u>*pivot*</u>) and partition S into
    - L elements <u>less than x</u>
    - E elements <u>equal x</u>
    - G elements <u>greater than x</u>
- Recur: sort L and G
- Conquer: join L, E and G
- *O(nlogn) runtime on average*



Example: Quick-Sort Tree



1. Sorting Methods

- Aim: to ***sort*** the vertices in *L2* into an order that minimizes # of crossings

(1) Adjacent-Exchange Method

- exchange adjacent pair of vertices using the crossing numbers, in a way similar to bubble sort
- Scan the vertices of *L2* from left to right, exchanging an adjacent pair *u, v* whenever $c_{uv} > c_{vu}$
- $O(|L2|^2)$ *time*

(1) Adjacent-Exchange

Algorithm 13: greedy_switch

repeat
  for $u := 1$ *to* $|V_2| - 1$ **do**
    if $c_{u(u+1)} > c_{(u+1)u}$ **then**
      switch vertices at positions u and $u + 1$;
until *the number of crossings was not reduced*;

(2) Split Method

(2) Split Method [Eades, Kelly 86]

- quick sort:
  - choose a *pivot vertex p* in *L2*
  - place each vertex u to the left of *p*, if $c_{up} < c_{pu}$,
  - place each vertex u to the right of *p*, otherwise.

- Apply recursively to the left & right of p

- O($|L2|^2$) time in worst case; O((|L2|log (|L2|) in practice

Algorithm 9.3 *Split*

Input: two-layered digraph $G = (L_1, L_2, E)$, an order x_1 for L_1

Output: vertex order x_2 for L_2

if L_2 is not empty **then**

(a) Choose a pivot vertex $p \in L_2$.

(b) $V_{left} = \emptyset$; $V_{right} = \emptyset$

(c) **foreach** vertex $u \in L_2$ such that $u \neq p$ **do**

  if $c_{up} \leq c_{pu}$
  then place u in V_{left}
  else place u in V_{right}

(d) Recursively apply the algorithm to the digraphs induced by V_{left} and V_{right}, and output the concatenation of the outputs of these two applications.

□

2. The Barycenter Method

- The most popular method
- x-coordinate of each vertex *u* in *L2* is chosen as the barycenter(average) of the x-coordinates of its neighbors
- *x2(u) = bary(u) = 1/deg(u) Σ x1(v), v* is a neighbor

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

- If two vertices have the same barycenter, then separate them arbitrary by a small amount
- Can be implemented in linear time

3. The Median Method

- x-coordinate of each vertex *u* in *L2* is chosen as the median of the x-coordinates of its neighbors
- *v1, v1, …, vj:* neighbors of *u* with *x1(v1)* < *x1(v2)* < … < *x1(vj)*
  - *med(u)* = *x1(vj/2)*
  - if *u* has no neighbor, then *med(u)* = 0

- How to use *med(u)* to order the vertices in *L2*: sort *L2* on *med(u)*
- If *med(u)* = *med(v)*
  - Place the odd degree vertex on the left of the even degree vertex
  - If they have the same parity, choose the order of *u* & *v* arbitrary

- Can be computed in time, using a linear-time median finding algorithm [Aho Hopcroft Ullman 83]

Analysis

[Theorem] if opt(G,x1)= 0, then bar(G,x1)=med(G,x1)=0

Performance guarantees

Theorem 1:

The *barycenter method* is at worst O(sqrt(n)) times optimal.

Theorem 2:

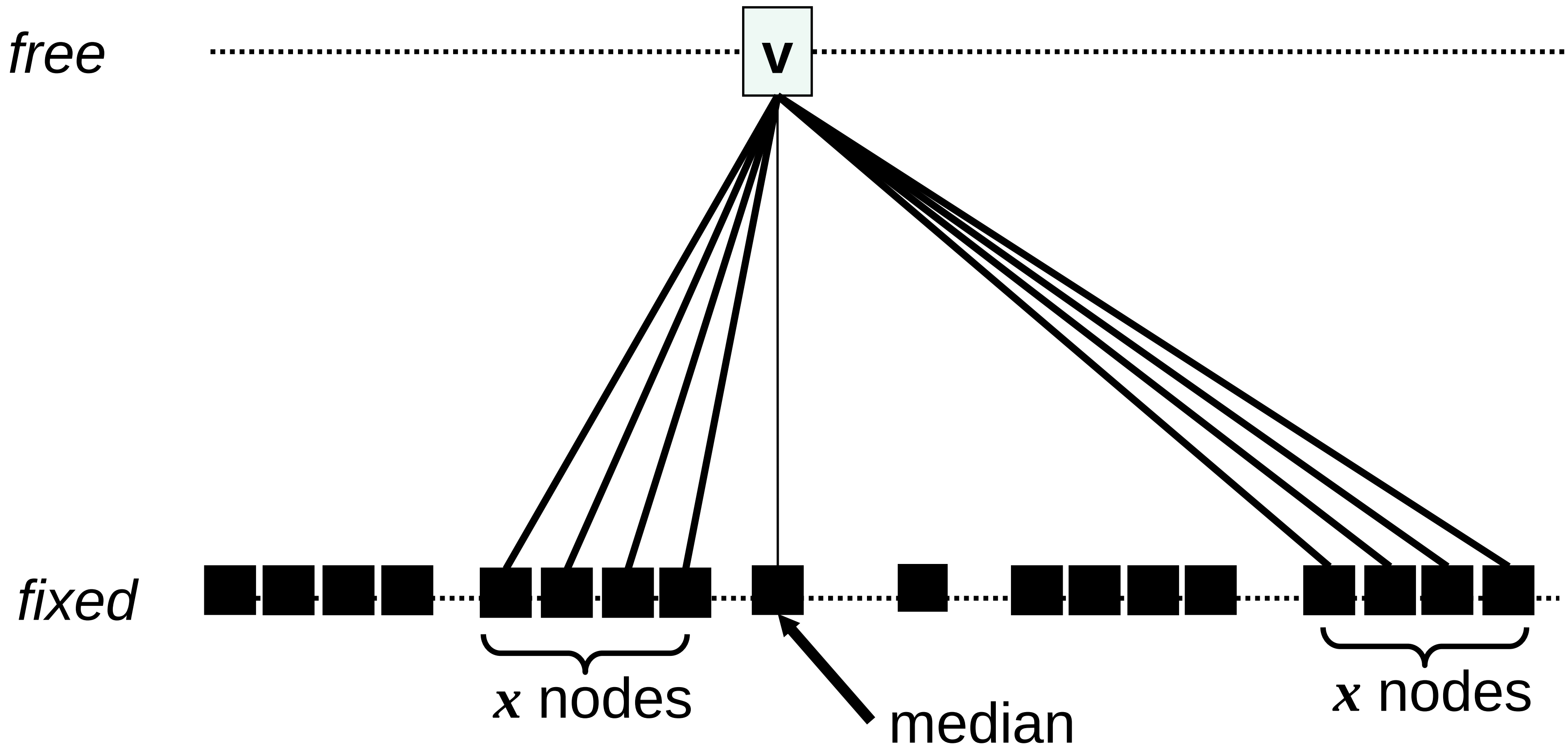
The *median method* is at worst 3 times optimal.

$$(1)\ \frac{bar(G)}{opt(G)}\ is\ O\left(\sqrt{n}\right)$$

$$(2)\ \frac{med(G)}{opt(G)} \leq 3$$

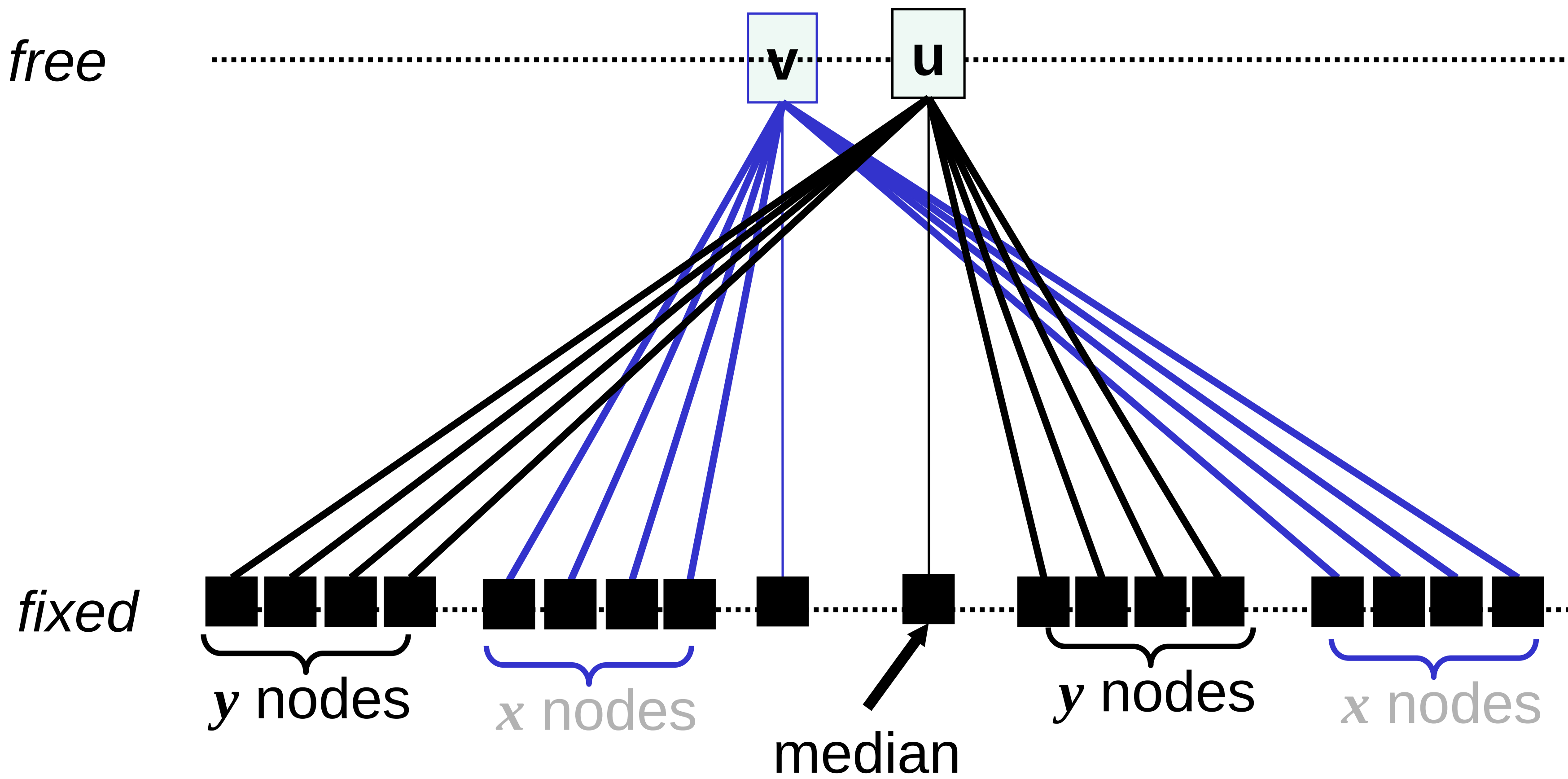
3. Median Method

- Some intuition behind Theorem 2
(median method is at worst 3 times optimal).



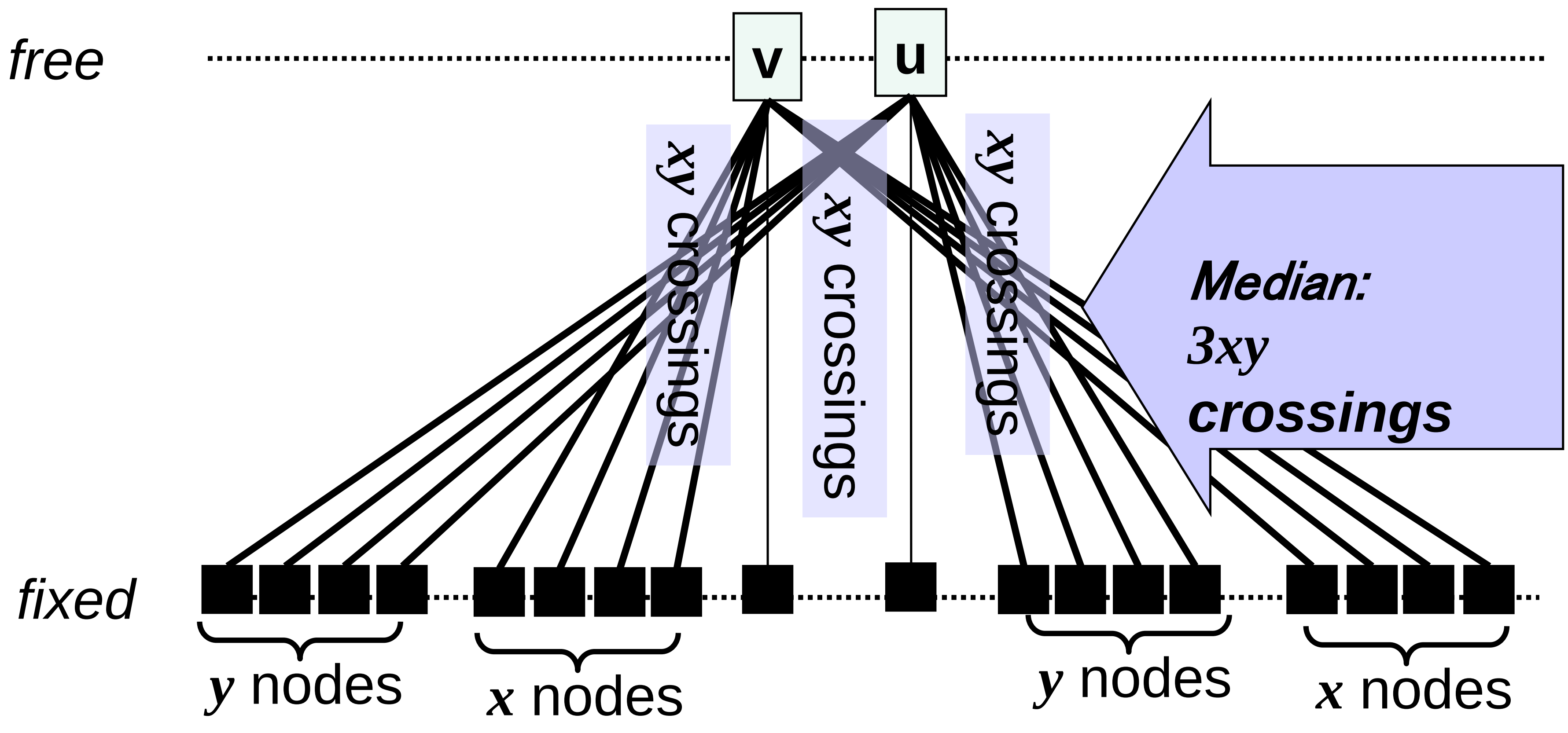
3. Median Method

- Some intuition behind Theorem 2 (median method is at worst 3 times optimal).



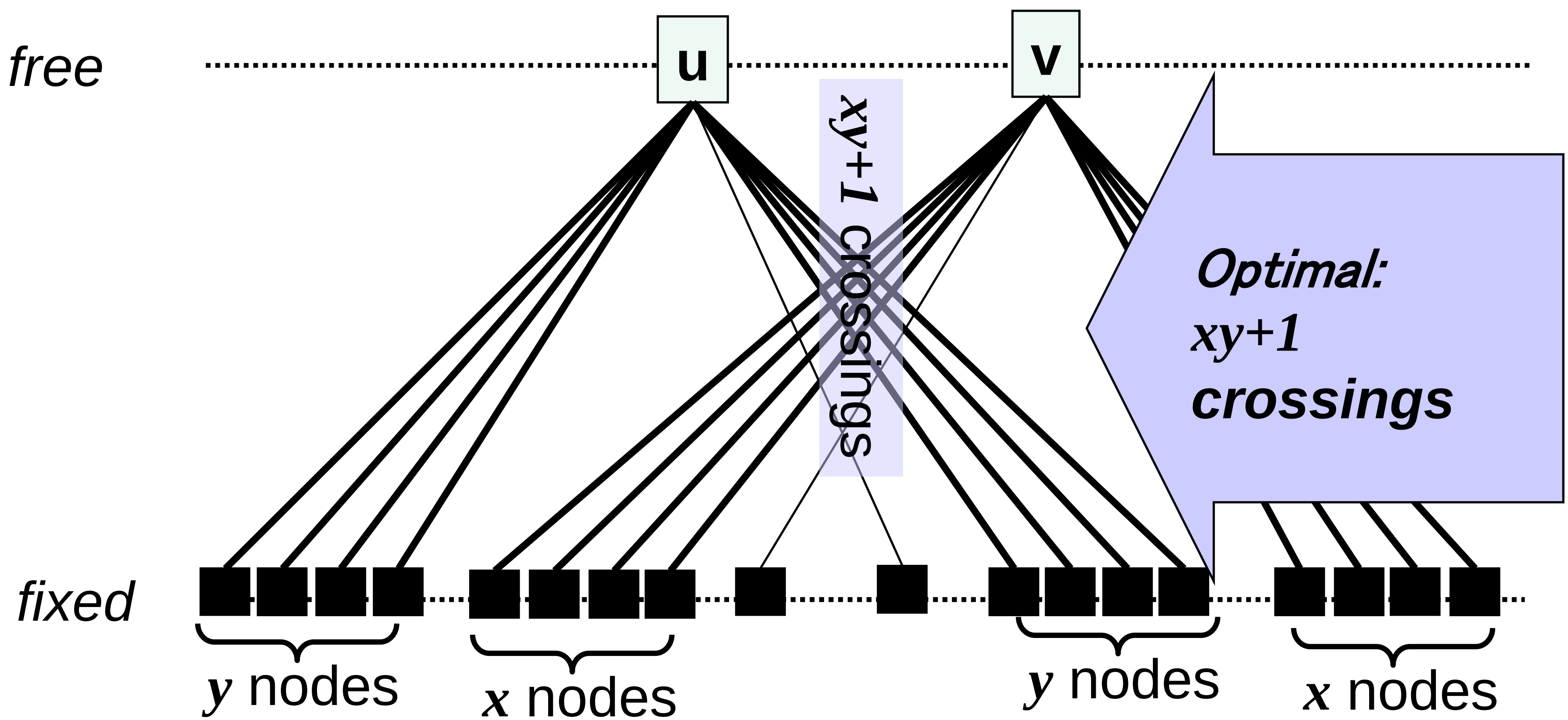
3. Median Method

- Median placement:



3. Median Method

- **Optimal placement:**



3. Median Method

- Median: at most $3xy$ crossings
- Optimal: at least $xy+1$ crossings

- *Theorem 2:* The *median method* is at worst 3 times optimal.

Best approximation algorithm:

[Nagamochi 05] 1.4664

4. Integer Programming methods

- Integer programming approach may be used for two-layer crossing problem

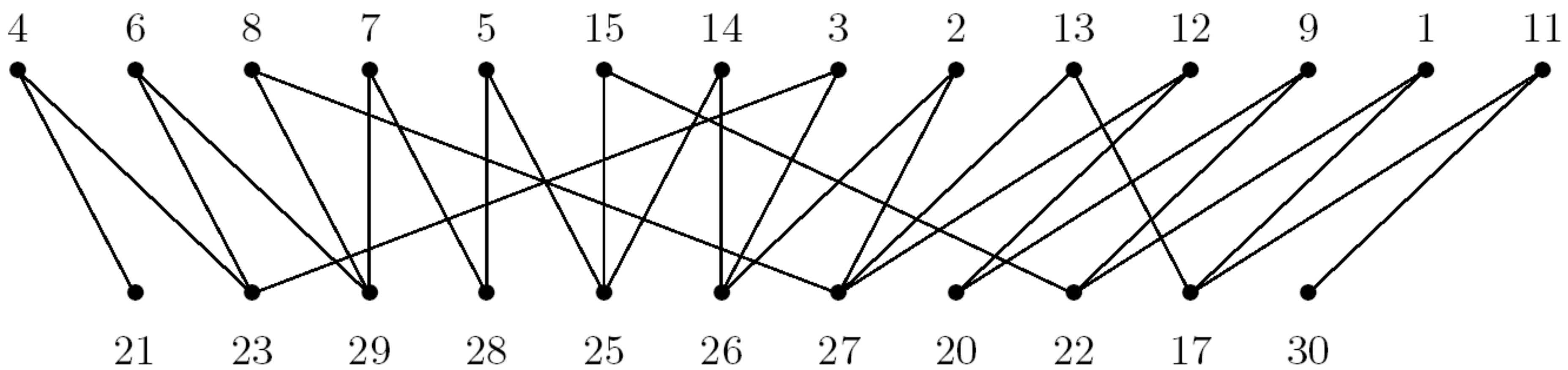
- Solving integer programs require sophisticated technique: branch and cut approach can be used to obtain an optimal solution for digraphs of limited size [Junger Mutzel 97]

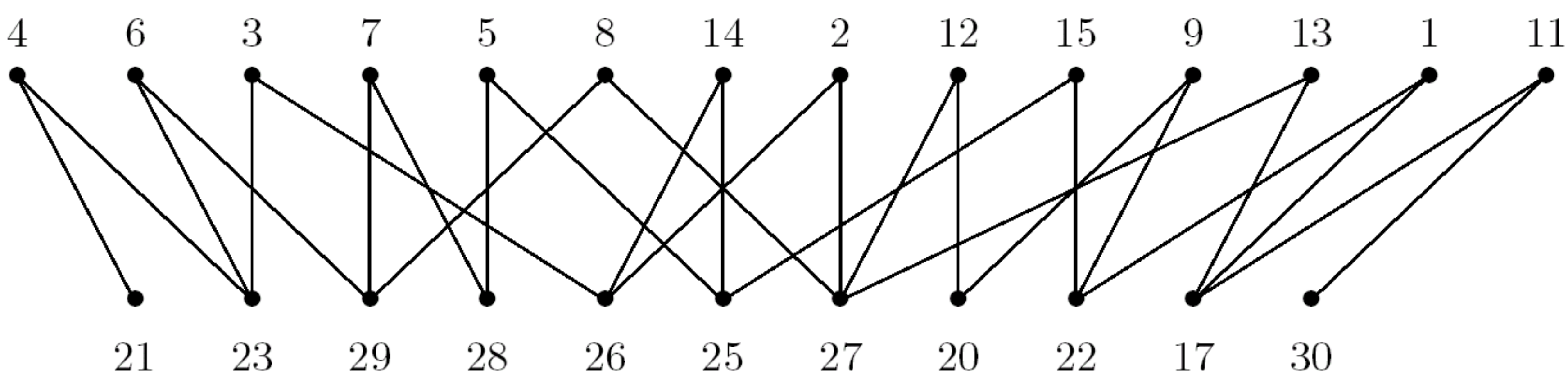
- Advantage: find the optimal solution
- Disadvantage: no guarantee to terminate in polynomial time

- Successful for small to medium sized digraphs

5. Planarization method [Mutzel97]

- Use <u>maximal planar subgraph</u> approach: NP-hard problem
- Integer linear programming [Mutzel 97]





Step 4. Horizontal Coordinate Assignment

Step 4. Horizontal Coordinate Assignment

- Bends occur at the dummy vertices in the layering step.
- We want to reduce the angle of such bends by choosing an x-coordinate for each vertex, without changing the ordering in the crossing reduction step

- Optimization problem with constraints
  - draw each directed path as straight as possible
  - ensure the ordering in each layer (enforce minimal distance)
- It may affect the width of the drawing
- Some layered drawing requires exponential area with straight lines

- Quadratic programming problems can be solved by standard methods, but it requires considerable computational resource

- [Brandes Kopf 02] simple fast heuristic

Extension

Radial Level Drawing
Extended Level Drawing
2.5D Level Drawing

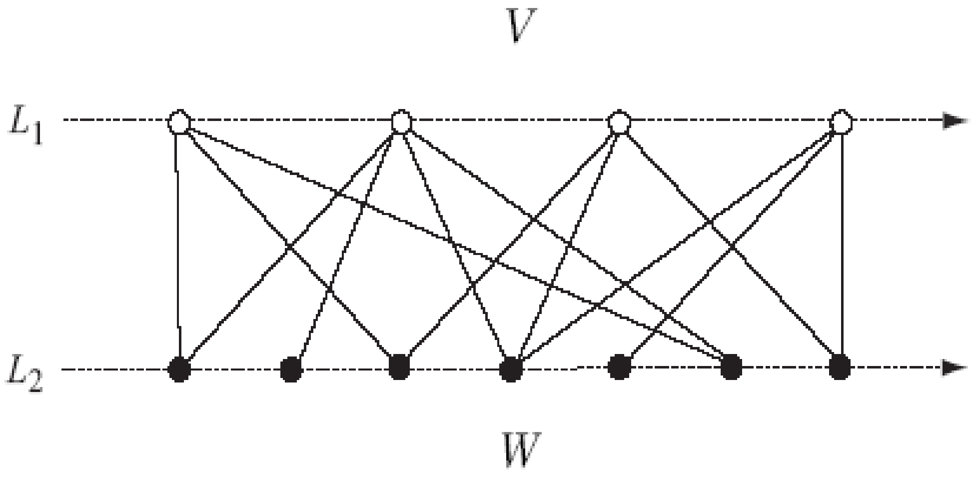
Radial Level Drawing (Radial Sugiyama)

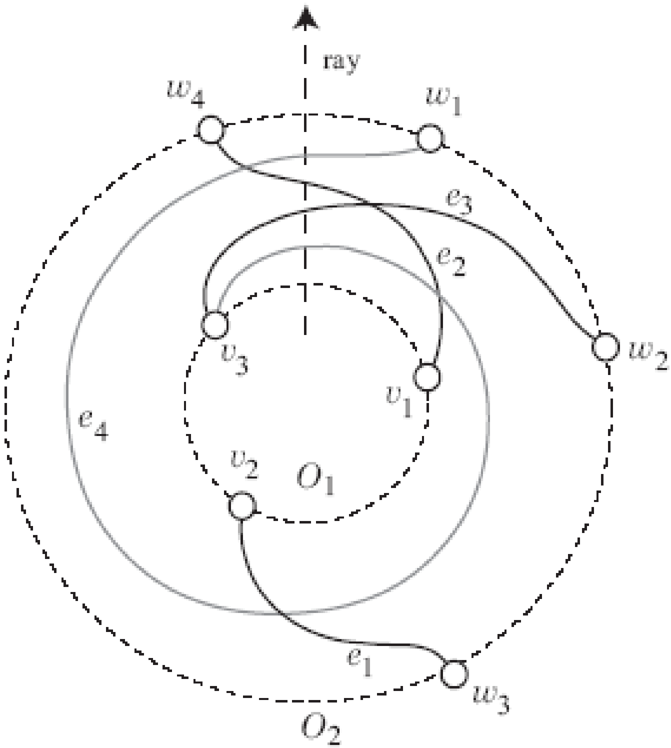
One-sided Radial Crossing Minimization: NP-hard

[Bachmaier 09] Sifting heuristic

[Hong, Nagamochi 09] 15-approximation algorithm

[Hong, Nagamochi 10] 4.3992-approximation for free layer without leaf node

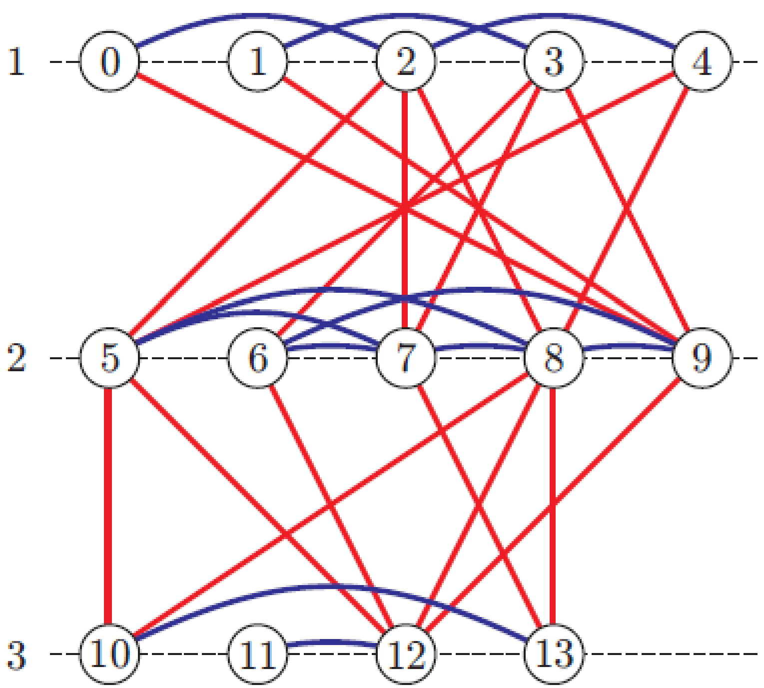


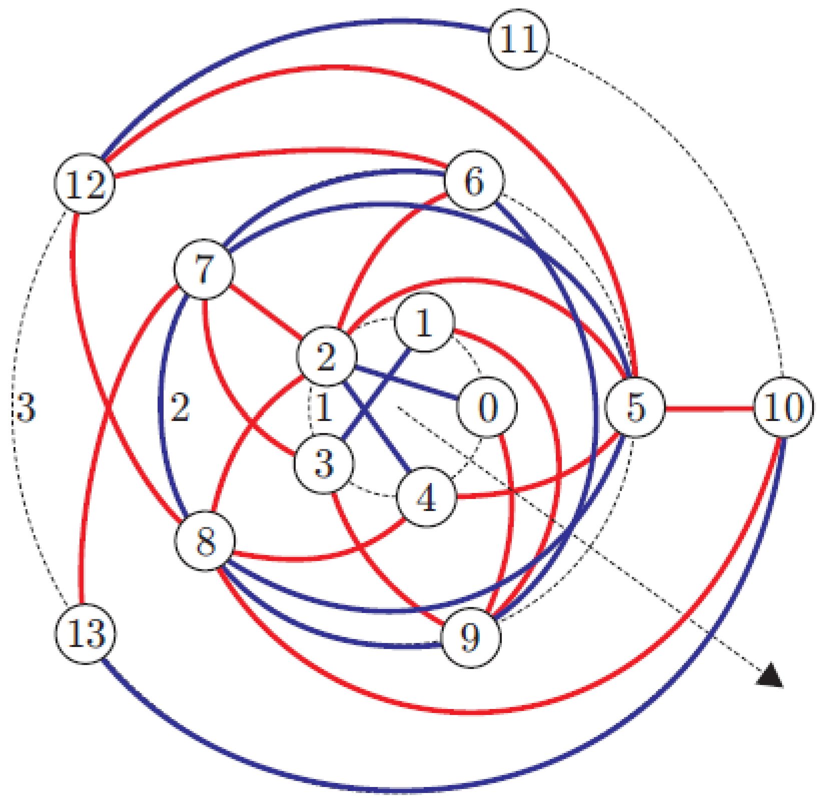


Extended Level Drawing

Crossing Minimization with <u>Inter-layer edges and Intra-layer edges</u>: NP-hard

[Bachmaier,Buchner,Forster,Hong 10] radial sifting heuristics



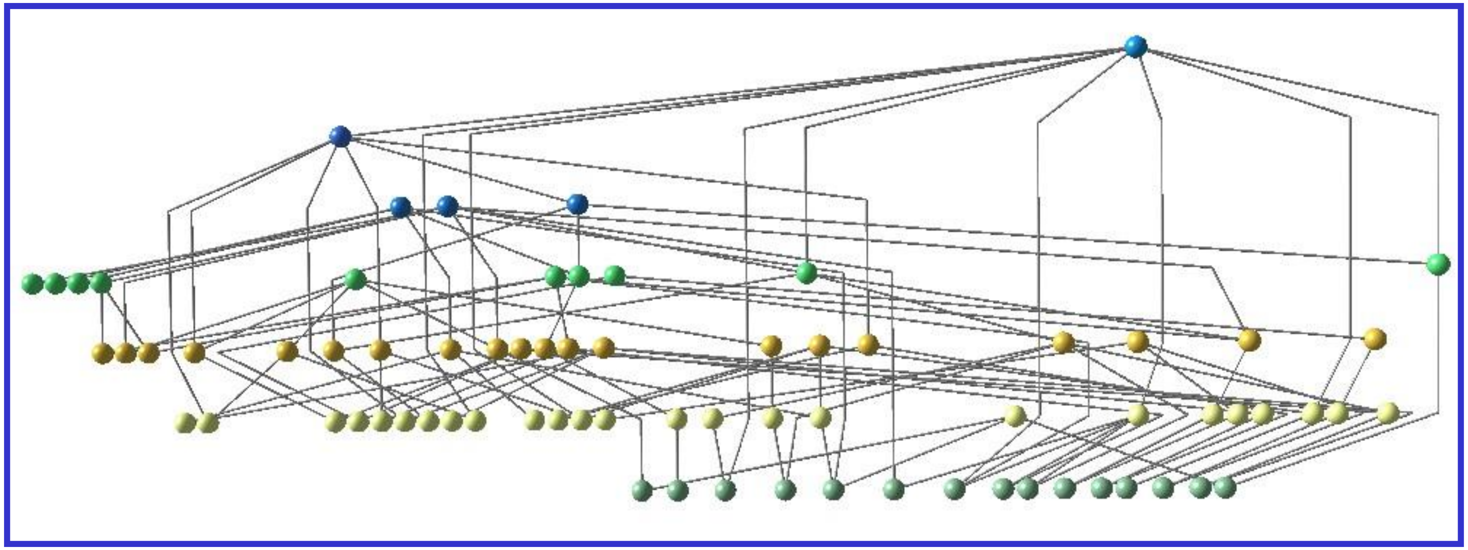


2.5D Sugiyama Method [Hong, Nikolov 06/07]

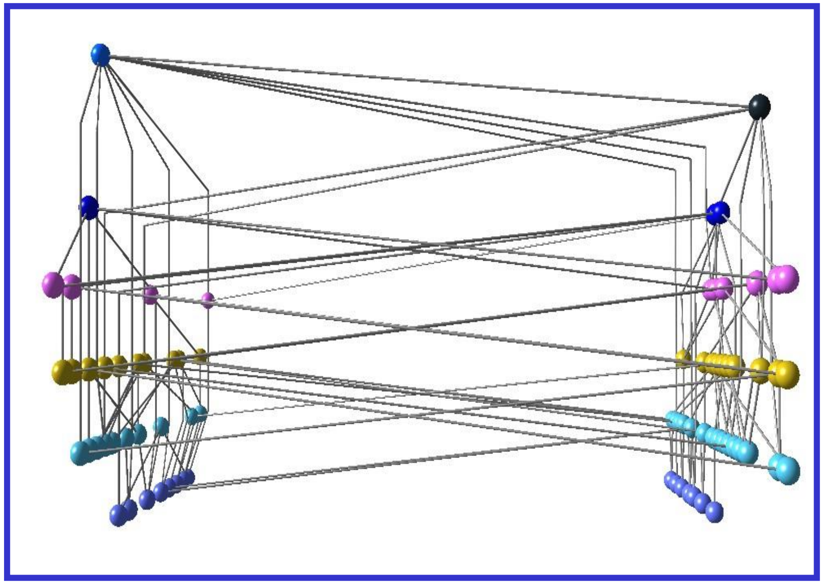
5 steps:

- step 0. *Wall assignment: divide G into k subgraph & draw each subgraph on a 2D plane (wall)*
- step1. Cycle removal: make acyclic digraph
- step2. Layer assignment: assign y-coordinates
- step3. Crossing reduction: determine the order of vertices in each layer
- step4. Horizontal coordinate assignment: assign x-coordinates

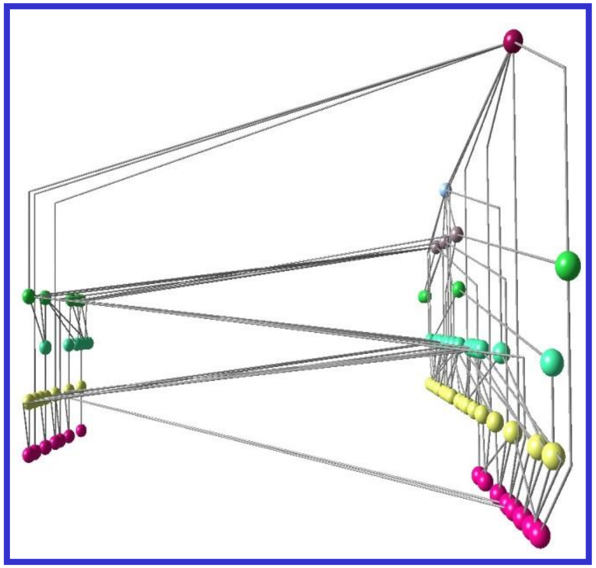
Wall Assignment Algorithm



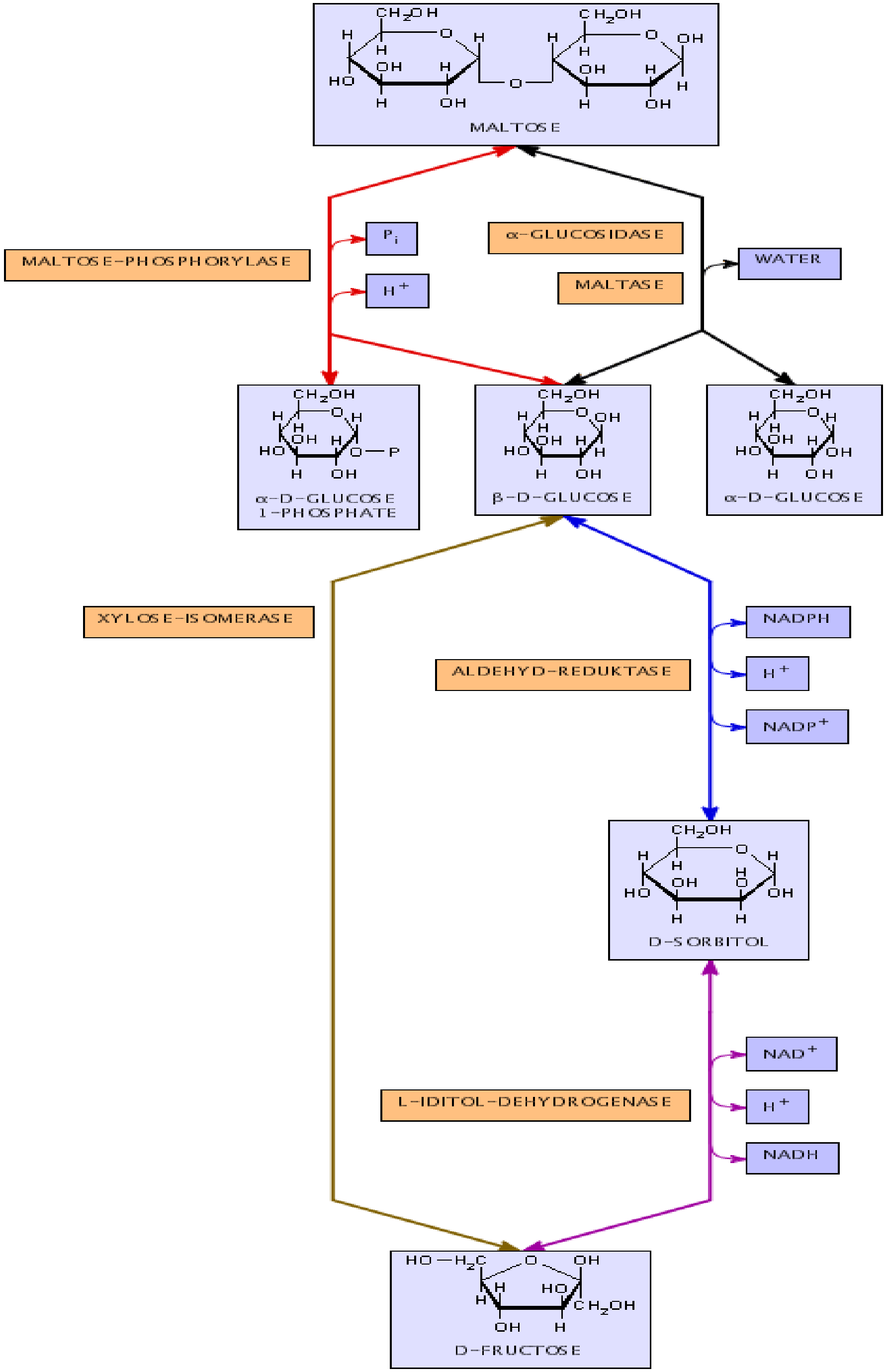
2D



Balanced Min-Cut



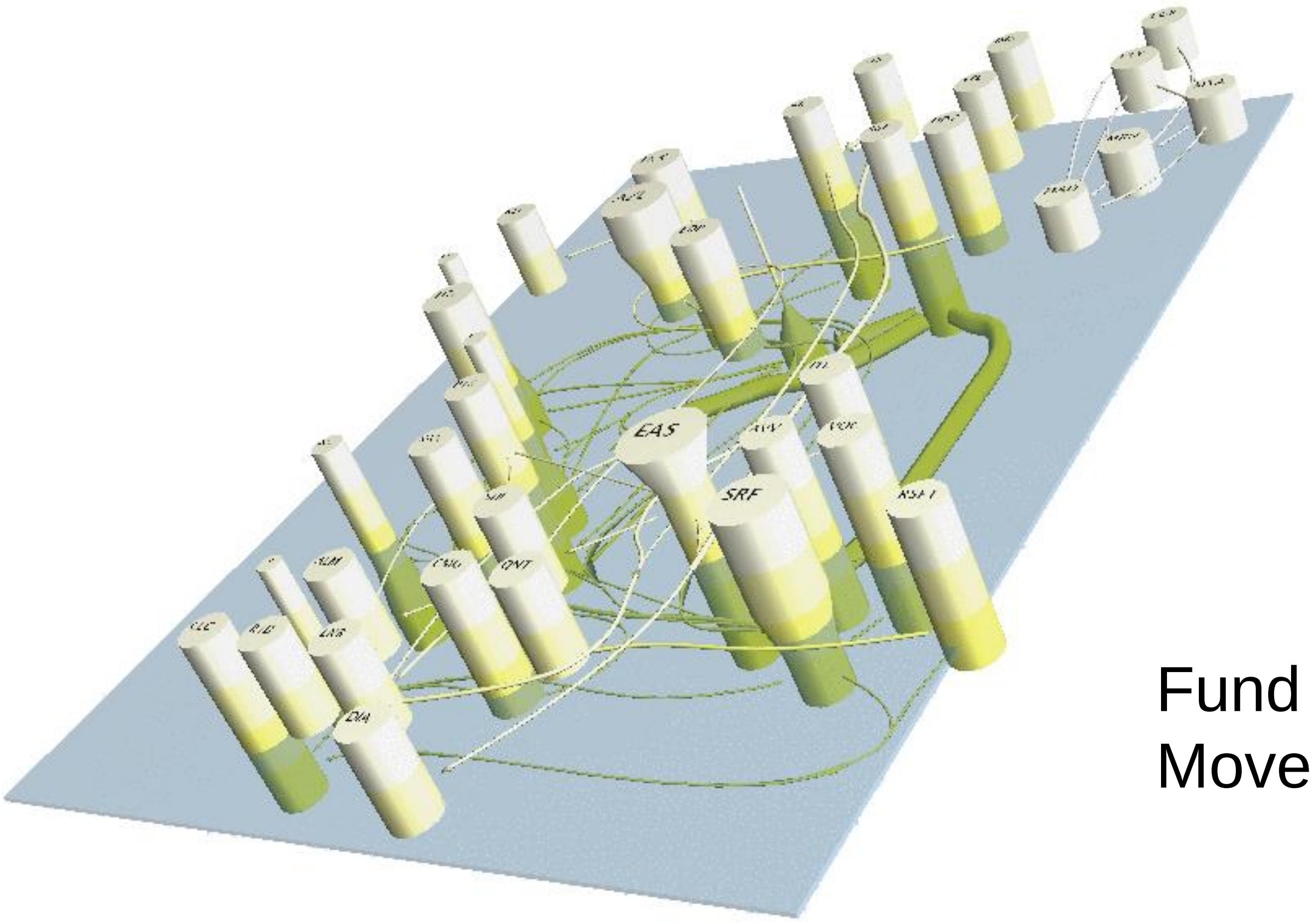
Zig-Zag



Visualization of Biochemical Pathways

Falk Schreiber

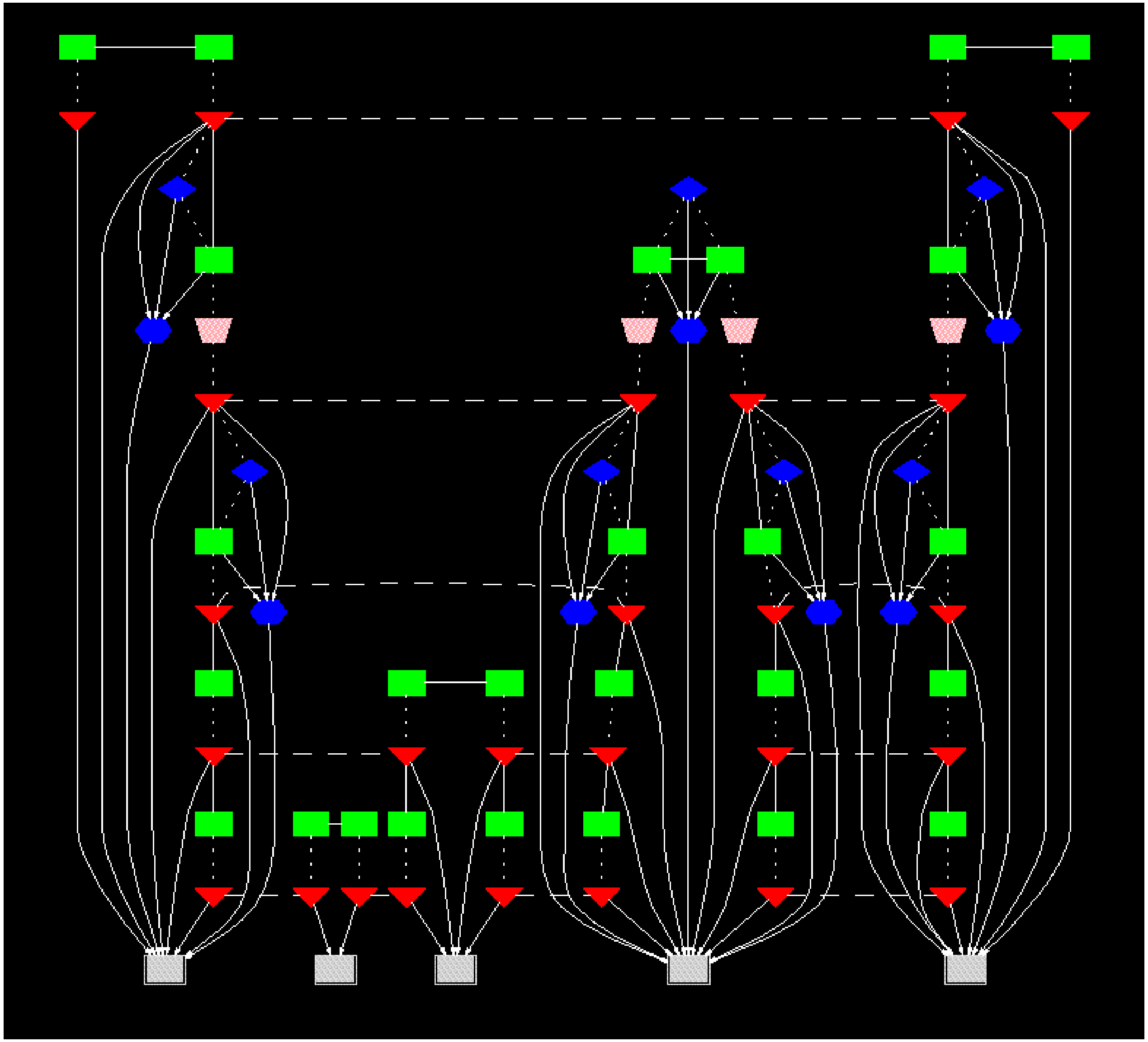
Visualisation of Stock Market Data



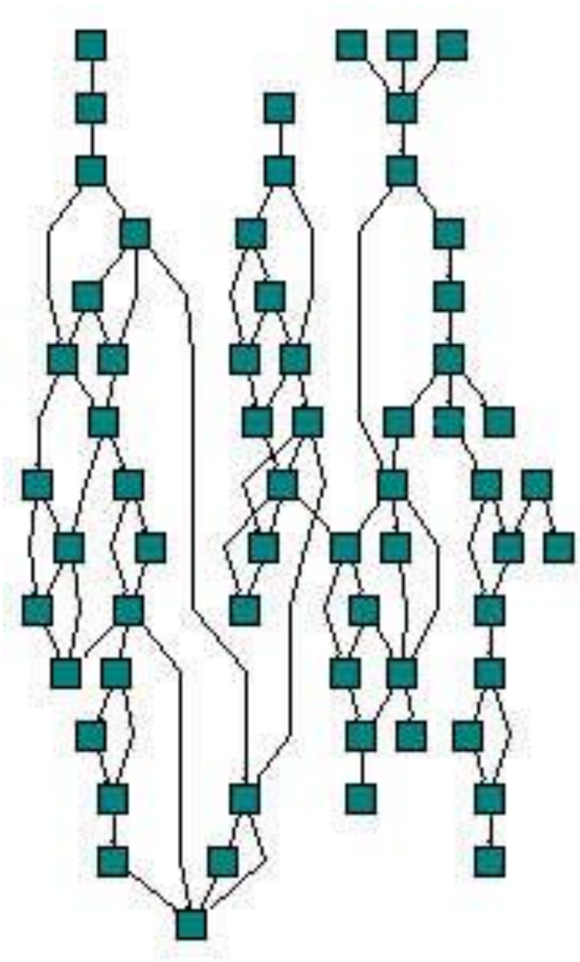
Fund Manager
Movement Graph

Tim Dwyer

GraphViz from AT&T

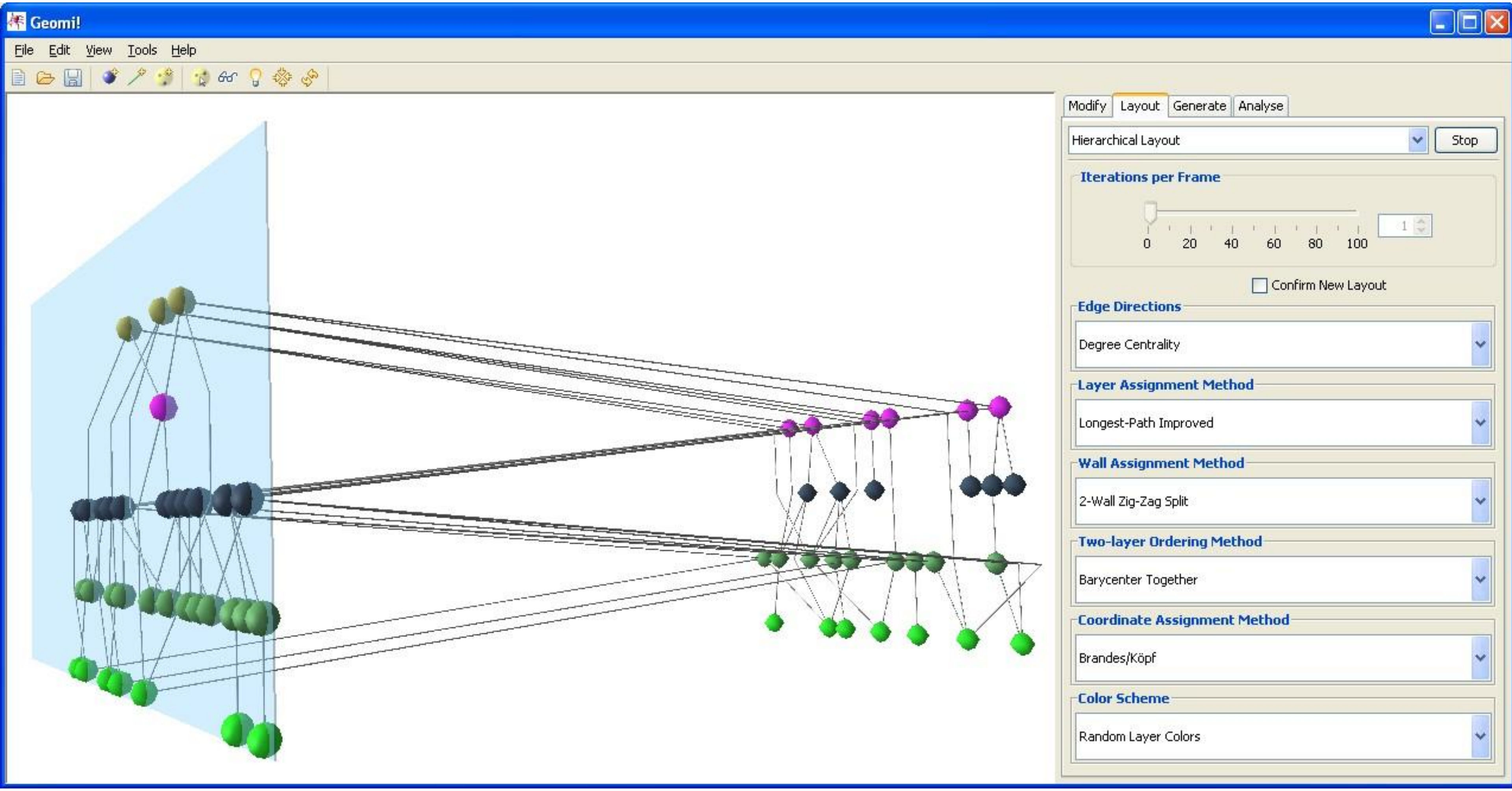


Tom Sawyer Software, USA



Hierarchical layout

GEOMI: 2.5D Hierarchical Layout



Summary

- Sugiyama Method (Layered/Level Drawing) is the most common method to visualise directed graphs
- 4 Steps: each step involved NP-hard problems for optimisation
- There are many good heuristics available for each step
- well used in industy and many applications
- more difficult to implement than spring algorithm/force directed methods

Assignment 1 (10 marks)

choose 2 graphs, and create good visualisation

Sep 13 THU 5pm

Assignment 2

- Form a group (up to 6) by week 4 at Canvas
- Initial Report (5 marks): Sep 20 THU 5pm