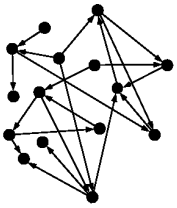
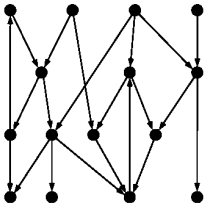
Layered Graph Drawing (Sugiyama Method)

Drawing Conventions and Aesthetics



■ a digraph



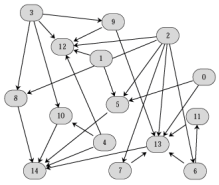
■ A possible layered drawing

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.

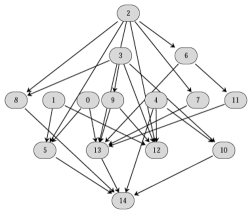
The Sugiyama method

- Layered networks are often used to represent dependency relations.
- Sugiyama *et al.* developed a simple method for drawing layered networks in 1979.
- Sugiyama's aims included:
  - few edge crossings
  - edges as straight as possible
  - nodes spread evenly over the page

The Sugiyama method

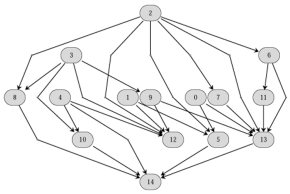


The Sugiyama method



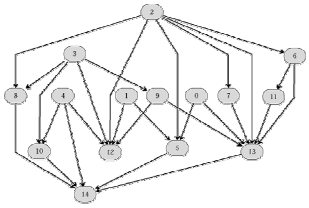
- Step 1
  Cycle Removal
- Step 2
  Layering

The Sugiyama method



- Step 1
  Cycle Removal
- Step 2
  Layering
- Step 3
  Node ordering

The Sugiyama method



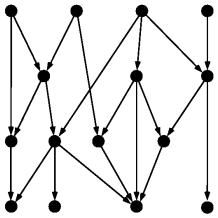
- Step 1
  Cycle Removal
- Step 2
  Layering
- Step 3
  Node ordering
- Step 4
  Coordinate assignment

Layered Drawing of Digraphs

- Polyline drawings of digraphs with vertices arranged in horizontal layers
  - Sugiyama, Tagawa and Toda '81
  - Eades and Sugiyama '91
  - Eades, Lin and Tamassia '95
  - Magnetic field approach of Sugiyama and Misue '95
  - Evolutionary algorithm of Branke et al. '01
  - Branch and cut algorithm of Healy and Nikolov '02
- Attractive in practice: most graph drawing systems include the Sugiyama method.

Step 1. Cycle Removal

- Input graph may contain cycles
  1. make an acyclic digraph by ***reversing*** some edges
  2. draw the acyclic graphs
  3. render the drawing with the original edge directions



- Acyclic graph by reversing two edges

Step 1. Cycle Removal

- Each cycle must have at least one edge against the flow
  - We need to keep the number of edges against the flow small
- Main problem: how to choose the set of edges R so that it is small
- Feedback arc set:
  - set of edges R whose removal makes the digraph acyclic
- Feedback arc set problem
  - find a minimum set E_f such that the graph(V, E' \ E_f) contains no cycles : NP-hard
- Maximum acyclic subgraph problem
  - find a maximum set E_a in E such that the graph (V, E_a) contains no cycles : NP-hard

Step 1. Cycle Removal

- Edges in E \ E_a will be removed
- Assume no 2-cycles (or delete both two edges)
- Heuristics
  1. Fast heuristic
  2. Enhanced Greedy heuristic

– Randomized algorithm [Berger and Shor 90]: O(mn) expected time
– Exact algorithm: [Grotschel et al '85, Reinelt '85]

A. Fast heuristic

- Maximum acyclic subgraph problem
  - equivalent to unweighted linear ordering problem: find an ordering o of the vertices such that the # of edges (u,v), $o(u) > o(v)$ is minimized.
- Easiest heuristic
  - Take an arbitrary ordering
  - Then delete the edges (u,v) with $o(u) > o(v)$
  - May use given ordering: BFS or DFS
  - No performance guarantee: reverse |E|-|V|-1 edges (DFS)
- Heuristic that guarantees acyclic set of size at least ½|E|
  - Take the better of an arbitrary ordering and its reverse
  - Same ratio is also achieved by a random ordering of the vertices

B. Enhanced greedy heuristic

- Greedy cycle removal heuristic [Eades et al 93]
  - Sources & sinks play a special role: edges incident to sources & sinks cannot be part of any cycle
  - Iteratively remove vertices from G
  - Greedy: choice of vertices to be removed

Greedy Cycle Removal

- Greedy Cycle Removal [Eades et al 93]
  - If there exists a source or a sink, then remove it and add all its incident edges to E_a
  - Otherwise, choose a vertex u such that ***|outdeg(u)-indeg(u)|*** is maximized; remove it and all its incident edges to E_a
  - performance

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

  - Can be implemented in linear time and space
  - Sparse graph: E_a with at least 2/3|E| edges

Greedy Cycle Removal

Algorithm 9: An Enhanced Greedy Heuristic

```
E_a = ∅;
while G is not empty do
1   while G contains a sink v do
        add δ⁻(v) to E_a and delete v and δ⁻(v) from G;
2   delete all isolated vertices from G;
3   while G contains a source v do
        add δ⁺(v) to E_a and delete v and δ⁺(v) from G;
4   if G is not empty then
        let v be a vertex in G with maximum value |δ⁺(v)| − |δ⁻(v)|;
        add δ⁺(v) to E_a and delete v and δ(v) from G;
```

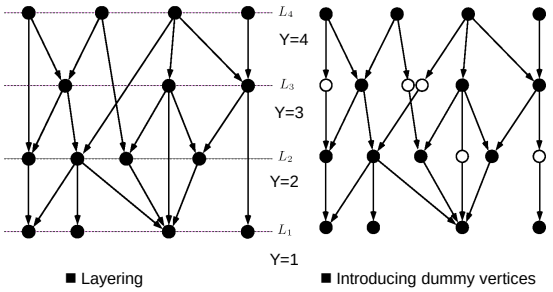
Analysis

- Delete all 2-cycles before applying Greedy-cycle-removal
- [Theorem] Let G be a connected digraph with n vertices & m edges and without 2-cycles.
  Greedy-Cycle-Removal computes an acyclic subgraph of G with at least m/2 + n/6 edges.
- [Theorem] Greedy-Cycle-Removal can be implemented in linear time & space
- Simple & speedy
- Sparse graph [EL95]
  [Theorem] Let G be a connected digraph with n vertices & m edges and without 2-cycles. If each vertex of G has total degree at most 3, then Greedy-Cycle-Removal computes an acyclic subgraph of G with at least 2m/3 edges.

Step 2. Layer Assignment

- Layering: partition V into $L_1, L_2, \ldots, L_h$
- Layered (di)graph: digraph with layers
- Height h: # of layers
- h-layered graph: digraph with height h
- Width w: # of vertices with largest layer
- Span of an edge
- Proper digraph: no edge has a span > 1

- Some application, vertices are preassigned to layers
- However, in most applications, we need to transform an acyclic digraph into a layered digraph



■ Layering ■ Introducing dummy vertices

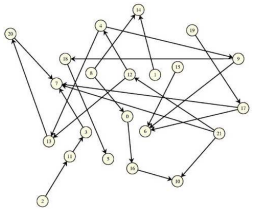
Step 2. Layer Assignment

- 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
     - C. the number of dummy vertices on an edge measures the y extent of the edge: avoid long edges
- Methods
  - A. Longest path layering: minimize height
  - B. Layering to minimize width
  - C. Minimize the number of dummy vertices

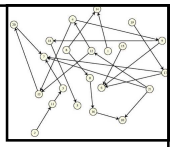
Three Layering Algorithms

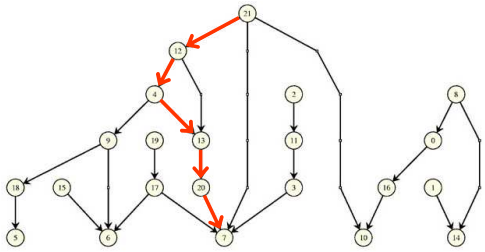
- Longest Path
- Coffman-Graham
- Network Simplex

Grafo1012 (Di Battista *et al., Computational Geometry: Theory and Applications*, (7), 1997)



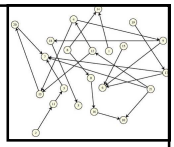
Longest Path Layering

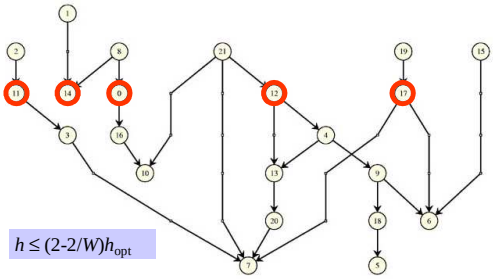




Coffman-Graham Layering

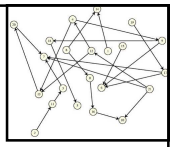
(1972)

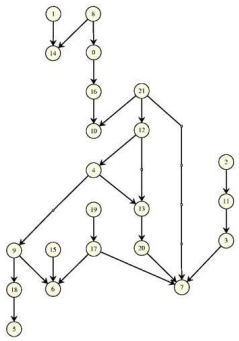




$h \le (2\text{-}2/W)h_{opt}$

Network Simplex Layering (AT&T, 1993)





ILP formulation

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

$$y(u) \in \mathbb{Z}, \forall u \in V,$$
$$y(u) \ge 1, \forall u \in V,$$
$$y(u) - y(v) \ge 1, \forall (u,v) \in E.$$

A. Longest path layering

- Minimizing 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

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

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

- Can be computed in linear time
- Main drawback: too wide

B. Layering to minimize width

- Finding a layering with minimum height subject to a maximum width constraint: ***Precedence-constrained multiprocessor scheduling problem*** -> NP-complete [Garey & Johnson '79]
- Coffman-Graham Layering
  - Input: reduced graph G (no transitive edges) and w
  - Output: layering of G with width at most w
  - Aim: ensure the height of the layering is kept small [LS77]
  - Two phases
    1. Order the vertices
    2. Assign layers
- Width: does not count dummy vertices

Coffman-Graham Layering

- Simple lexicographic order:

$S \prec T$ 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 $S \setminus \{\max(S)\} \prec T \setminus \{\max(T)\}$

- First phase: lexicographical ordering
- Second phase: ensure that no layer receive more than w vertices
- [LS77] height is not too large $h \leq (2 - \frac{2}{w})h_{opt}$

Coffman-Graham Layering

Algorithm 12: Coffman-Graham-Algorithm

```
foreach v ∈ V do π(v) := n + 1;
for i = 1 to |V| do
    choose a vertex v with π(v) = n + 1
                and minimum set {π(u) | (u, v) ∈ E} with respect to ≺;
    π(v) := i;
k := 1; L_1 := ∅; U := V;
while U ≠ ∅ do
    choose u ∈ U such that every vertex in {v | (u, v) ∈ E} is in V \ U
                                                and π(u) is maximized;
    if |L_k| < w and N+(u) ⊆ L_1 ∪ L_2 ∪ ... L_{k-1} then
        add u to L_k;
    else
        k := k + 1; L_k := {u};
    delete u from U;
```

C. Minimizing # of dummy vertices

- one can compute a layering in polynomial time that minimizes the number of dummy vertices [Gansner, Koutsofios, North & Vo '93]
- f = $\Sigma_{(u,v)\in V}$ (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
- Integer linear programming problem

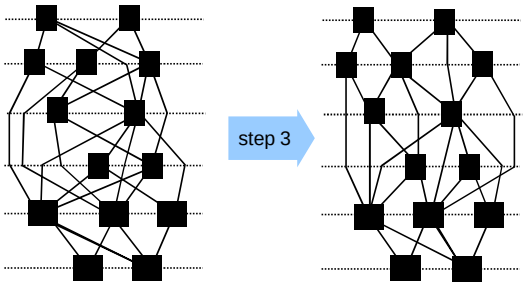
Remark

- Methods
  1. Minimizing the height: Longest path layering
  2. Layering with given width: Coffman-Graham algorithm: width is more important than height
  3. Minimizing the total edge span (# of dummy vertices) : relatively compact drawing

Step 3. Crossing Reduction

- 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 (combinatorial, rather than geometric)
- NP-complete, even for only two layers [Garey & Johnson '83]
- Many heuristics
  - Layer-by-layer sweep: two layer crossing problem
  - A. Sorting
  - B. Barycenter method
  - C. Median method
  - D. Integer programming method: exact algorithm

Crossing Reduction: ordering



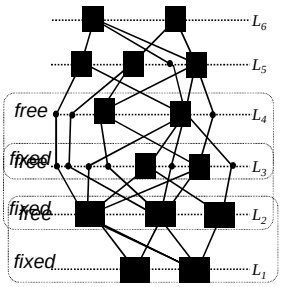
21 edge crossings

5 edge crossings

Layer-by-layer sweep

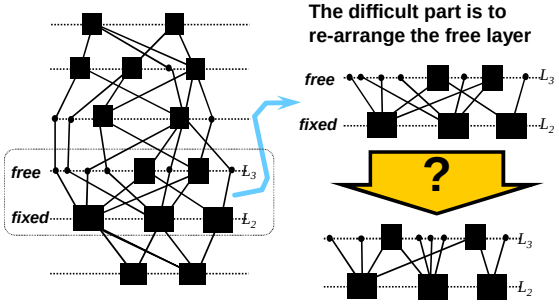
- A vertex ordering of layer L_1 is chosen
- For i = 2, 3, …, h
  - The vertex ordering of L_{i-1} is fixed
  - Reordering the vertices in layer L_i to reduce edges crossings between L_{i-1} and L_i
- Two layer crossing problem:
  given a fixed ordering of layer L_{i-1}, choose a vertex ordering of layer L_i to minimize # of crossings
- Several variations: layer-by-layer sweep

Layer-by-layer sweep



- Step 3 uses a "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.

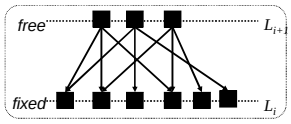
Two layer crossing problem



The difficult part is to re-arrange the free layer

Two layer crossing problem

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



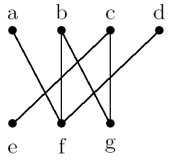
- *Heuristics*
  1. ***Barycenter method:*** place each free node at the barycenter of its neighbours.
  2. ***Median method:*** place each free node at the median of its neighbours.

Two layer crossing problem

- Given a two-layered digraph G=(L1,L2,E) and an ordering x1 of L1, find an ordering x2 of L2, such that cross(G,x1,x2) = opt(G,x1)
  - two-layered digraph G=(L1, L2, E)
  - cross(G, x1, x2): # of crossings in a drawing of G
  - opt(G,x1): $\min_{x2}$ cross(G, x1, x2)
- NP-complete: [Eades & Whitesides '94]
- Simple observation: let u and v be vertices in L2; then the # of crossings between edges incident to u and edges incident to v depends only on the relative positions of u and v and not on the position of the other vertices

Crossing number

- Crossing number c_{uv}
  - # of crossings that edges incident to u make with edges incident v, when $x_2(u) < x_2(v)$
  - # of pairs (u,w), (v,z) of edges with $x_1(z) < x_1(w)$



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

One-sided crossing minimization

$$\mathrm{opt}(G, \pi_1) = \min_{\pi_2} \mathrm{cross}(G, \pi_1, \pi_2)$$

Given a bipartite Graph $G = (V_1, V_2, E)$ and a permutation π_1 of V_1. Find a permutation π_2 of V_2 that minimizes the edge crossings in the drawing of G, i.e., $\mathrm{cross}(G, \pi_1, \pi_2) = \mathrm{opt}(G, \pi_1)$.

$$\mathrm{cross}(G, \pi_1, \pi_2) = \sum_{\pi_2(u) < \pi_2(v)} c_{uv} = \sum_{i=1}^{n_2 - 1} \sum_{j=i+1}^{n_2} c_{ij}$$

$$L = \sum_{\pi_2(u) < \pi_2(v)} \min\{c_{uv}, c_{vu}\}$$

A. Sorting Method

- Aim: to sort the vertices in L_2 into an order that minimizes # of crossings
- Naive algorithm: $O(|E|)^2$, can be reduced
- *Adjacent-Exchange*
  - exchange adjacent pair of vertices using the crossing numbers, in a way similar to bubble sort
  - Scan the vertices of L_2 from left to right, exchanging an adjacent pair u, v whenever $c_{uv} > c_{vu}$
  - $O(|L_2|^2)$ time
- *Split*
  - quick sort: choose a pivot vertex p in L_2, and place each vertex u to the left of p if $c_{up} < c_{pu}$, and to the right of p otherwise
  - Apply recursively to the left & right of p
  - $O(|L_2|^2)$ time in worst case; $O((|L_2| \log (|L_2|)$ in practice

Adjacent-Exchange

Algorithm 13: greedy_switch

```
repeat
    for u := 1 to |V2| − 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;
```

Split

Algorithm 14: split $(i, j : 1, \ldots, |V_2|)$

```
if j > i then
    pivot := low := i; high := j;
    for k := i + 1 to j do
        if c_{k pivot} < c_{pivot k} then
            π(k) := low; low := low + 1;
        else
            π(k) := high; high := high − 1;
    /* low == high */
    π(pivot) := low;
    copy π(i...j) into π2(i...j);
    split (i, low − 1);
    split (high + 1, j);
```

B. The Barycenter Method

- The most common method
- x-coordinate of each vertex u in L2 is chosen as the barycenter(average) of the x-coordinates of its neighbors
- $x_2(u) = \mathrm{bary}(u) = 1/\deg(u)\ \Sigma\ x_1(v)$, v is a neighbor of u

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

- If two vertices have the same barycenter, then order them arbitrarily
- Can be implemented in linear time

C. The Median Method

- Similar to the barycenter method
- x-coordinate of each vertex u in L2 is chosen as the median of the x-coordinates of its neighbors
- $v_1, v_2, \ldots, v_j$: neighbors of u with $x_1(v_1) < x_1(v_2) < \ldots < x_1(v_j)$
  - med(u) = $x_1(v_j/2)$
  - if u has no neighbor, then med(u) = 0
- How to use med(u) to order the vertices in L_2: sort L_2 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 arbitrarily
- Can be computed using a linear-time median finding algorithm [AHU83]

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:

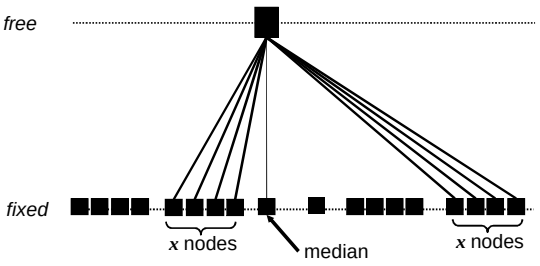
The *median method* is at worst 3 times optimal.

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

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

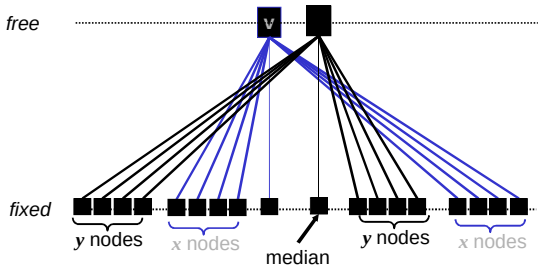
C. Median Method

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



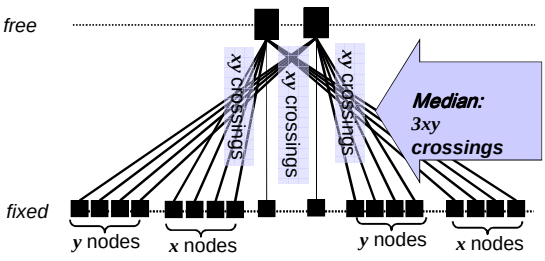
C. Median Method

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



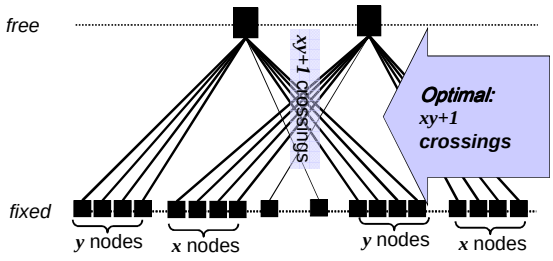
C. Median Method

- Median placement:



C. Median Method

- Optimal placement:



C. Median Method

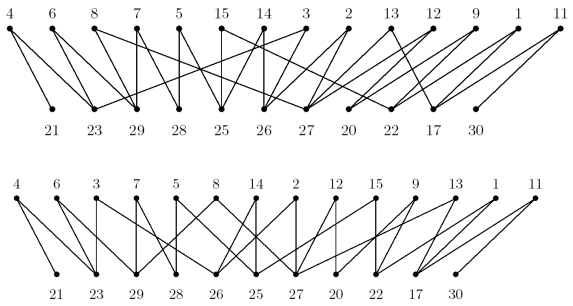
- **Median: at most $3xy$ crossings**
- **Optimal: at least $xy+1$ crossings**
- *Theorem 2:* **The *median method* is at worst 3 times optimal.**
- **In practice, there are many good methods, and the median is just one of them.**

D. 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 [JM97]**
- **Advantage: find the optimal solution**
- **Disadvantage: no guarantee to terminate in polynomial time**
- **Successful for small to medium sized digraphs**

E. Planarization method [Mutzel97]

- **Use maximal planar subgraph approach**



Remark

- **Median method seems very attractive**
- **Comparative tests**
  - **pseudo-random graphs [EK86, JM97]**
  - **real-world digraphs [GKNV93]**
  - **No single winner**
- **Use a hybrid approach**
  1. **Use the median method to determine the initial ordering**
  2. **Use an adjacent exchange method to refine**

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**

Priority Barycenter Method

- **Position vertices at the Barycenter of their neighbours**
  - **Can reuse "positions" from the ordering step**
- **Assign each vertex a priority**
  - **Priority = degree**
  - **Dummy vertices have the highest priority**
- **Enforce minimal distance between adjacent vertices**
  - **If two vertices are too close then move ONE of them to a safe distance**
  - **Move the vertex with the lower priority**