Graph Drawing Tutorial

Isabel F. Cruz

Worcester Polytechnic Institute

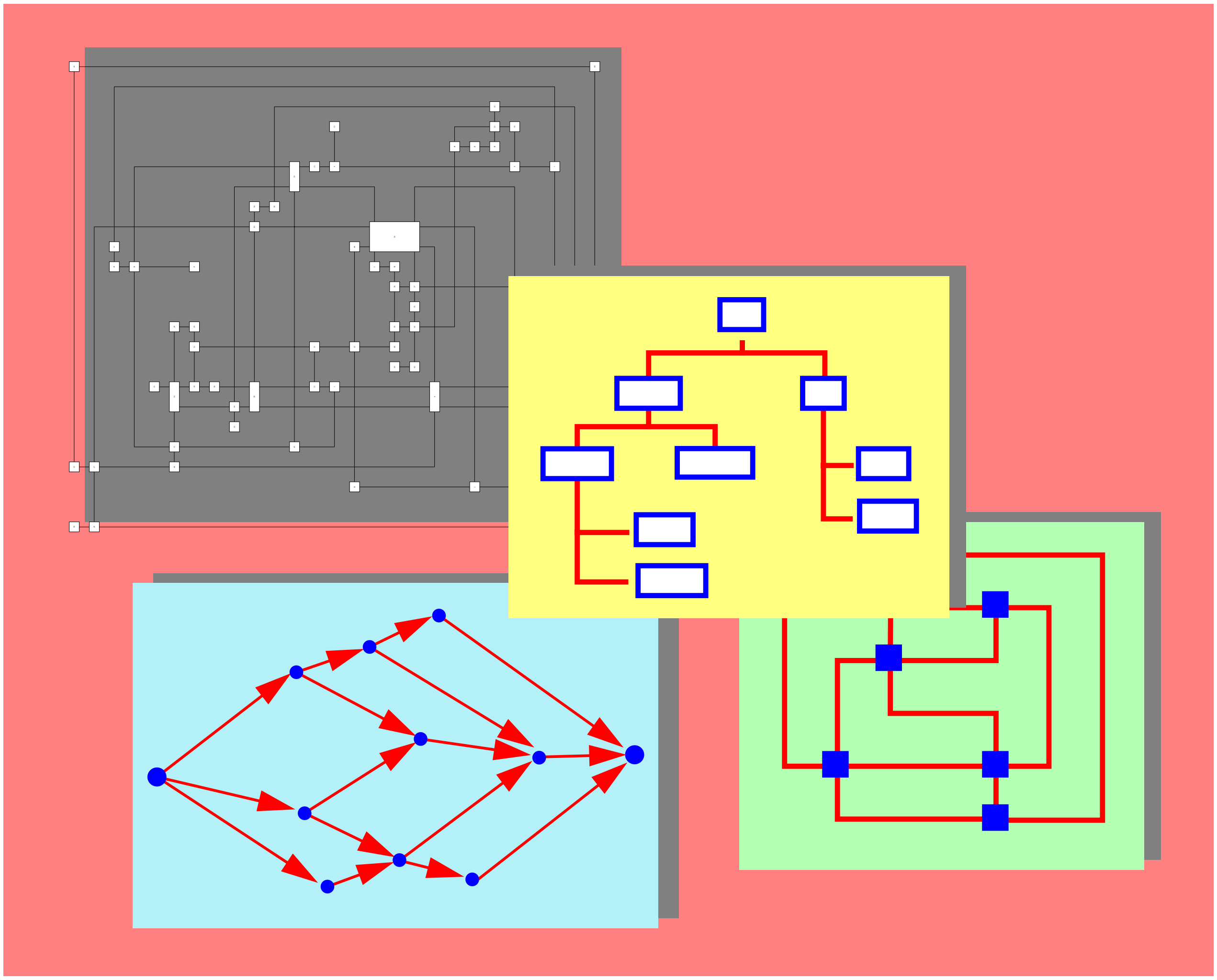
Roberto Tamassia

Brown University

Introduction

Graph Drawing

- models, algorithms, and systems for the visualization of ***graphs*** and ***networks***

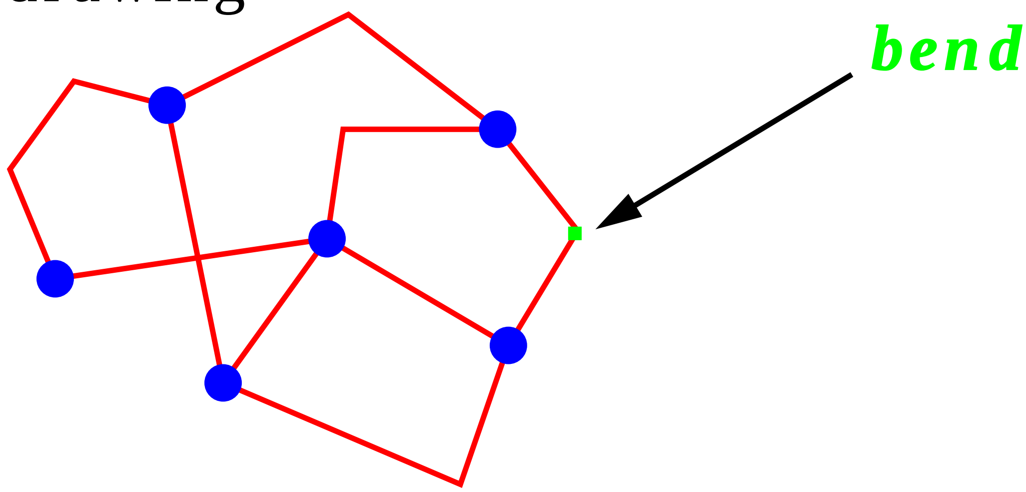


- applications to ***software engineering*** (class hierarchies), ***database systems*** (ER-diagrams), ***project management*** (PERT diagrams), ***knowledge representation*** (isa hierarchies), ***telecommunications*** (ring covers), ***WWW*** (browsing history) ...

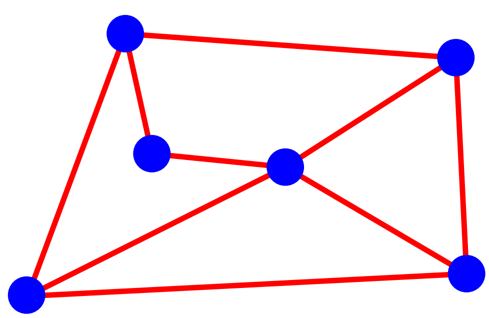
Drawing Conventions

- ***general constraints*** on the geometric representation of vertices and edges

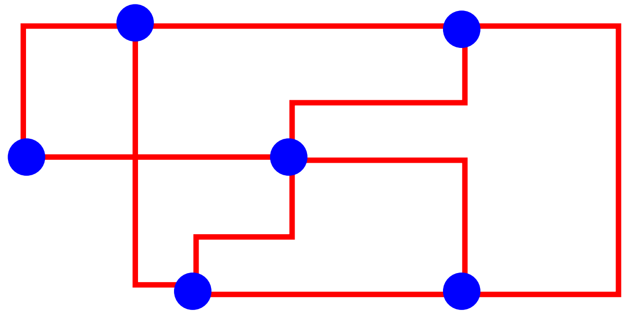
polyline drawing



planar straight-line drawing

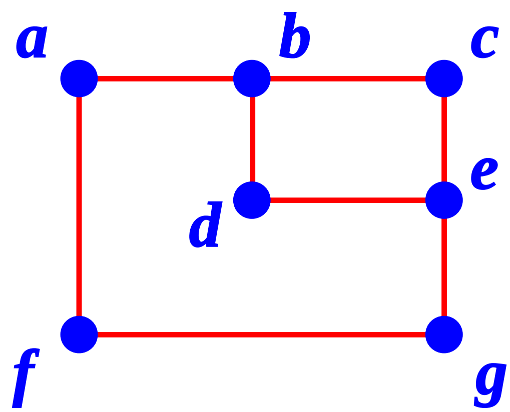


orthogonal drawing

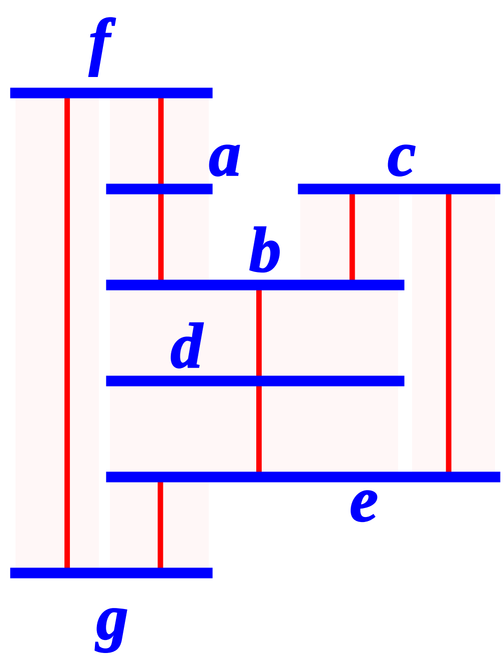


Drawing Conventions

planar othogonal straight-line drawing

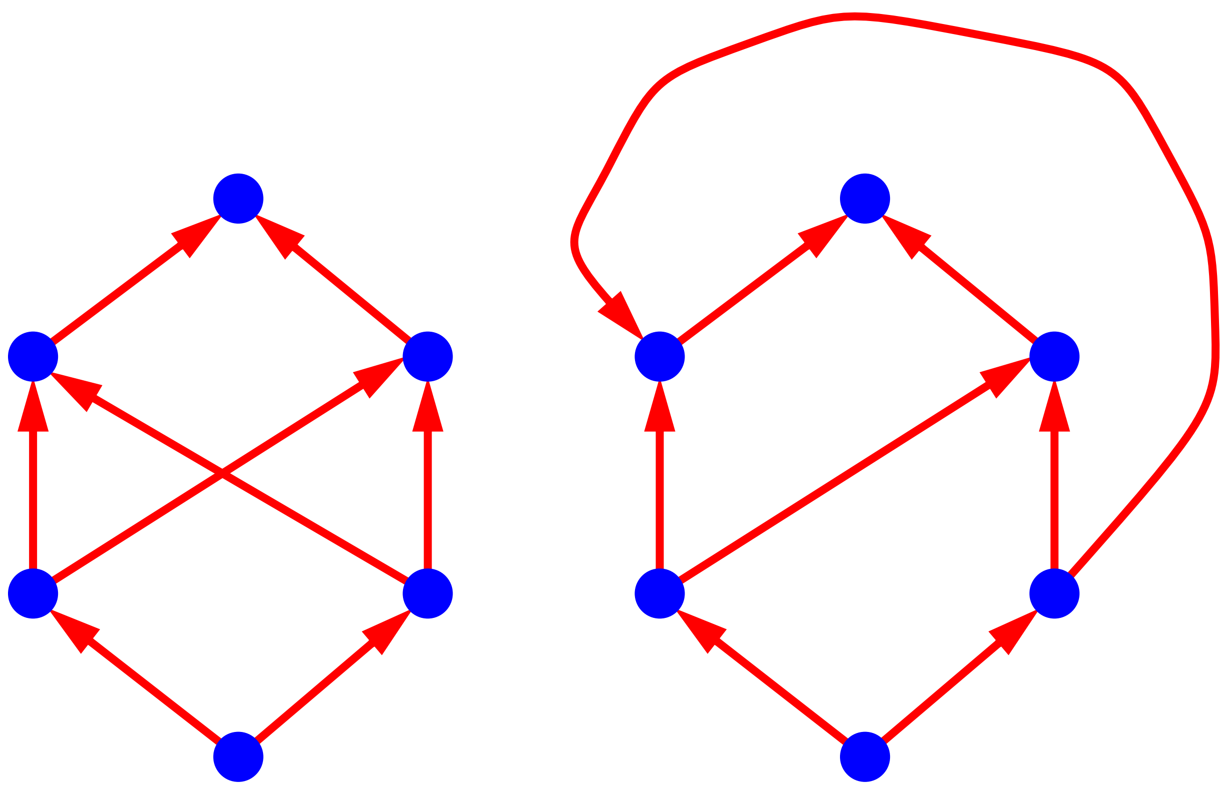


strong visibility representation



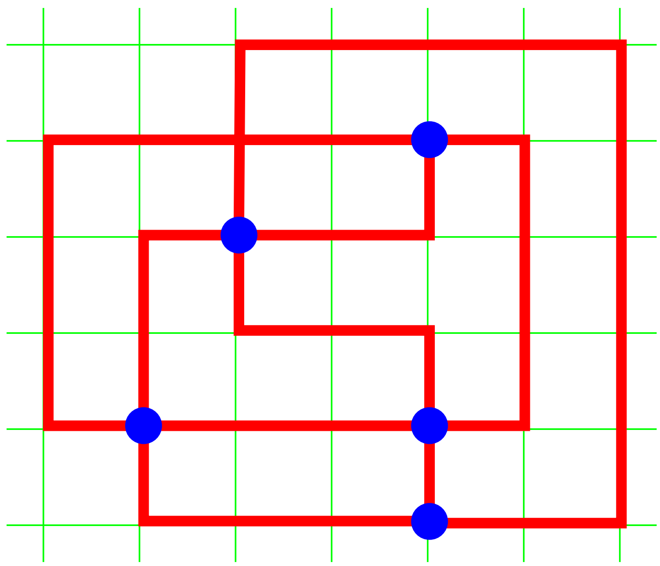
Drawing Conventions

- directed acyclic graphs are usually drawn in such a way that all edges “flow” in the same direction, e.g., from left to right, or from bottom to top
- such ***upward drawings*** effectively visualize hierarchical relationships, such as covering digraphs of ordered sets
- not every planar acyclic digraph admits a planar upward drawing



Resolution

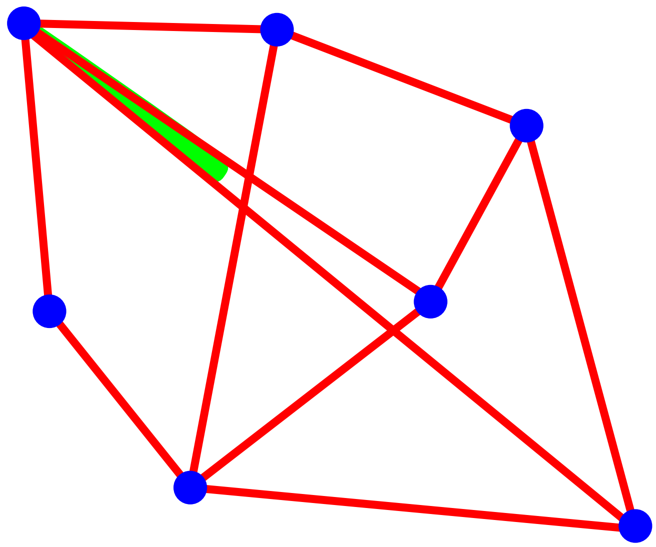
- display devices and the human eye have finite resolution
- examples of ***resolution rules***:
  - integer coordinates for vertices and bends (***grid*** drawings)



  - prescribed minimum distance between vertices
  - prescribed minimum distance between vertices and nonincident edges
  - prescribed minimum angle formed by consecutive incident edges (***angular resolution***)

Angular Resolution

- The ***angular resolution*** ρ of a straight-line drawing is the smallest angle formed by two edges incident on the same vertex



- ***High angular resolution*** is desirable in ***visualization*** applications and in the design of ***optical communication*** networks.
- A ***trivial upper bound*** on the angular resolution is

$$\rho \leq \frac{2\pi}{d}$$

where d is the maximum vertex degree.

Aesthetic Criteria

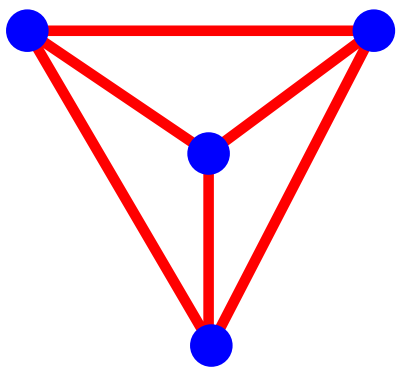
- some drawings are better than others in conveying information on the graph
- ***aesthetic criteria*** attempt to characterize readability by means of general ***optimization*** goals

Examples

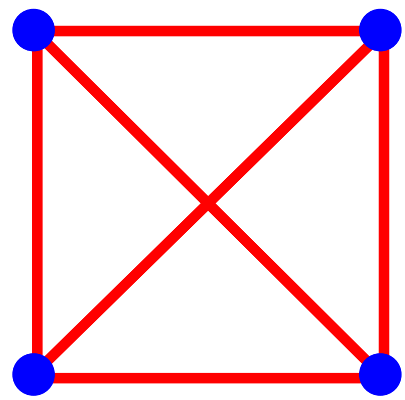
- minimize ***crossings***
- minimize ***area***
- minimize ***bends*** (in orthogonal drawings)
- minimize ***slopes*** (in polyline drawings)
- maximize ***smallest angle***
- maximize display of ***symmetries***

Trade-Offs

- in general, one cannot simultaneously optimize two aesthetic criteria



min # crossings

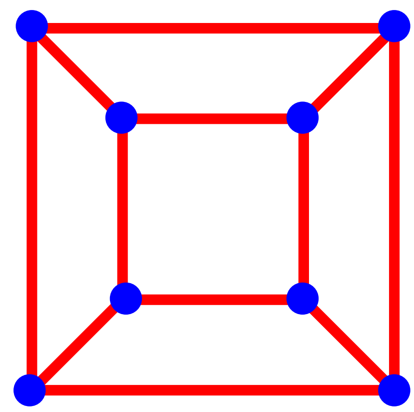


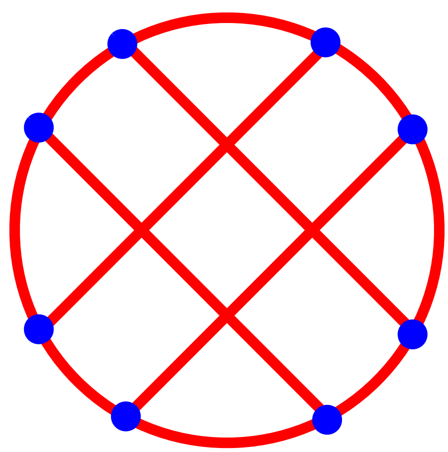
max symmetries

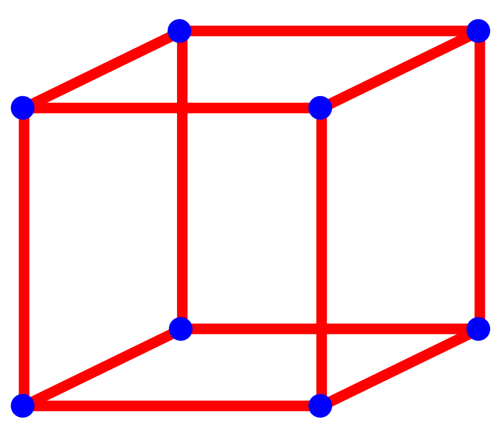
Complexity Issues

- testing planarity takes linear time
- testing upward planarity is NP-hard
- minimizing crossings is NP-hard
- minimizing bends in planar orthogonal drawing:
  - NP-hard in general
  - polynomial time for a fixed embedding

Beyond Aesthetic Criteria







Constraints

- some readability aspects require knowledge about the ***semantics*** of the specific graph (e.g., place "most important" vertex in the middle)
- ***constraints*** are provided as additional input to a graph drawing algorithm

Examples

- place a given vertex in the "middle" of the drawing
- place a given vertex on the external boundary of the drawing
- draw a subgraph with a prescribed "shape"
- keep a group of vertices "close" together

Algorithmic Approach

- Layout of the graph generated according to a prespecified set of aesthetic criteria
- Aesthetic criteria embodied in an algorithm as optimization goals. E.g.
  - minimization of crossings
  - minimization of area

Advantages

- Computational ***efficiency***

Disadvantages

- User-defined ***constraints*** are not naturally supported

Extensions

- A limited constraint-satisfaction capability is attainable within the algorithmic approach

  E.g., [Tamassia Di Battista Batini 87]

Declarative Approach

- Layout of the graph specified by a ***user-defined*** set of ***constraints***
- Layout generated by the ***solution*** of a ***system*** of constraints

Advantages

- ***Expressive power***

Disadvantages

- Some natural aesthetics (e.g., planarity) need ***complicated*** constraints to be expressed
- General constraint-solving systems are computationally ***inefficient***
- Lack of a powerful language for the specification of constraints (currently done with a detailed enumeration of facts, or with a set notation)

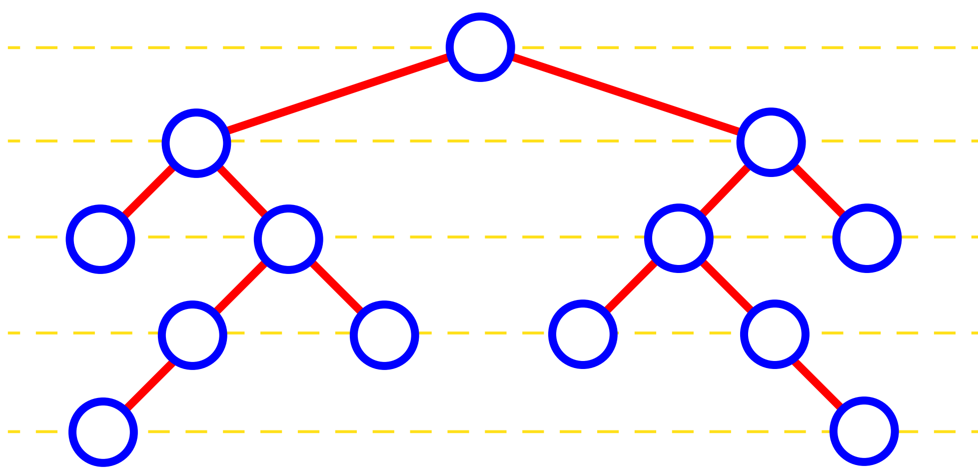
Getting Started with Graph Drawing

- Book on Graph Drawing by G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, ISBN 0-13-301615-3, *Prentice Hall*, (available in August 1998).
- Roberto Tamassia's WWW page
  http://www.cs.brown.edu/people/rt/
- Tutorial on Graph Drawing by Isabel Cruz and Roberto Tamassia (about 100 pages)
- Annotated Bibliography on Graph Drawing (more than 300 entries, up to 1993) by Di Battista, Eades, Tamassia, and Tollis. *Computational Geometry: Theory and Applications*, 4(5), 235-282 (1994).
- Computational Geometry Bibliography
  www.cs.duke.edu/~jeffe/compgeom/biblios.html
  (about 8,000 BibTeX entries, including most papers on graph drawing, updated quarterly)
- Proceedings of the Graph Drawing Symposium (Springer-Verlag, LNCS)
- Graph Drawing Chapters in:
  CRC Handbook of Discrete and Computational Geometry
  Elsevier Manual of Computational Geometry

Trees

Drawings of Rooted Trees

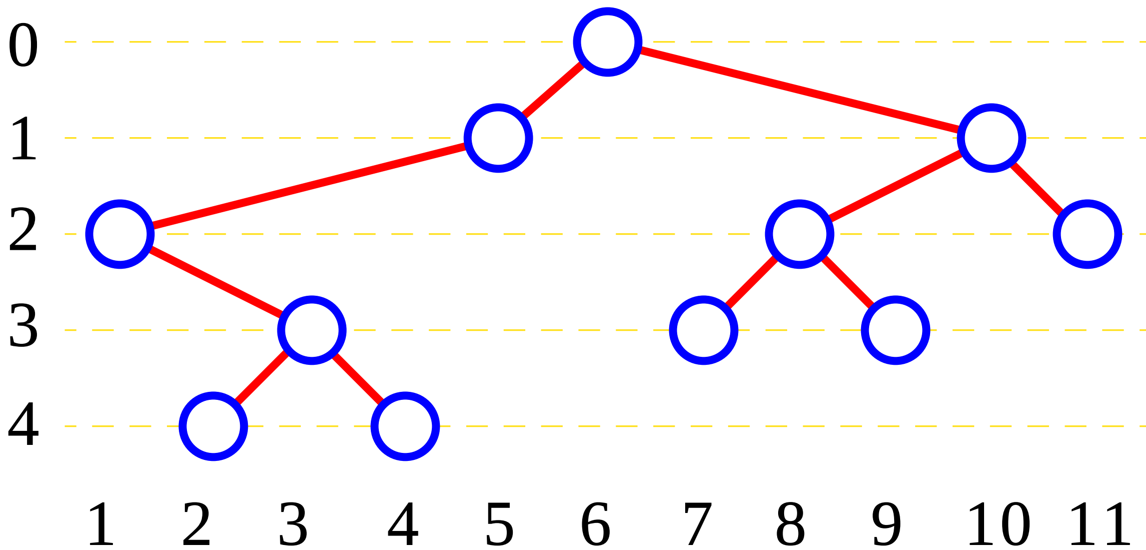
- the usual drawings of rooted trees are ***planar***, ***straight-line***, and ***upward*** (parents above children)
- it is desirable to minimize the ***area*** and to display ***symmetries*** and ***isomorphic subtrees***
- ***level drawing***: nodes at the same distance from the root are horizontally aligned



- level drawings may require $\Omega(n^2)$ ***area***

A Simple Level Drawing Algorithm for Binary Trees

- y(*v*) = distance from root
- x(*v*) = inorder rank

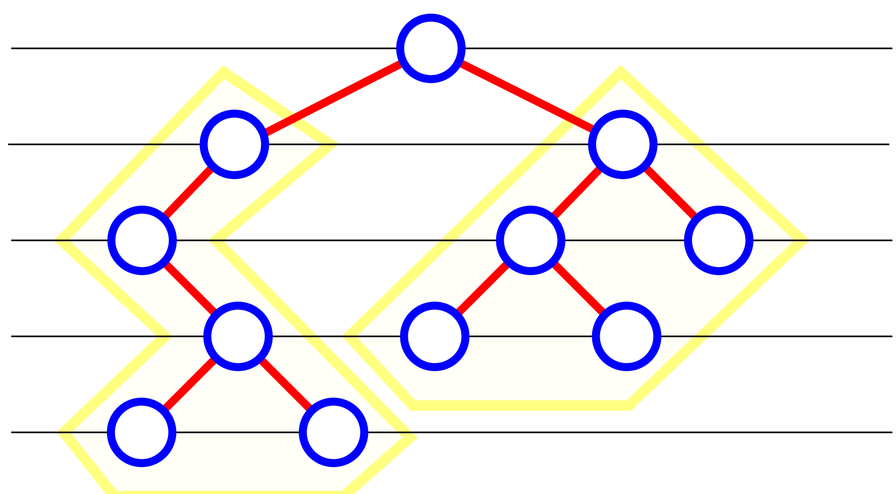


- level grid drawing
- display of symmetries and of isomorphic subtrees
- parent in between left and right child
- parents not always centered on children
- width = n – 1

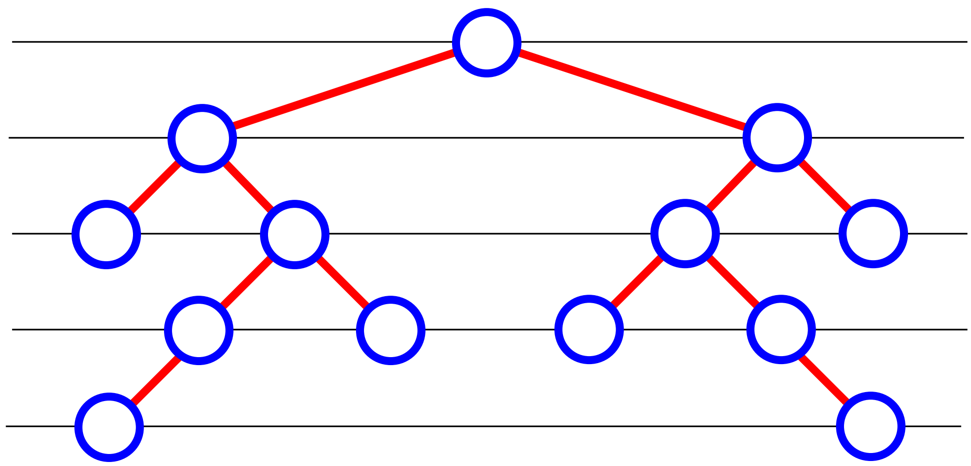
A Recursive Level Drawing Algorithm for Binary Trees

[Reingold Tilford 1983]

- draw the left subtree
- draw the right subtree
- place the drawings of the subtrees at horizontal distance 2
- place the root one level above and half-way between the children
- if there is only one child, place the root at horizontal distance 1 from the child

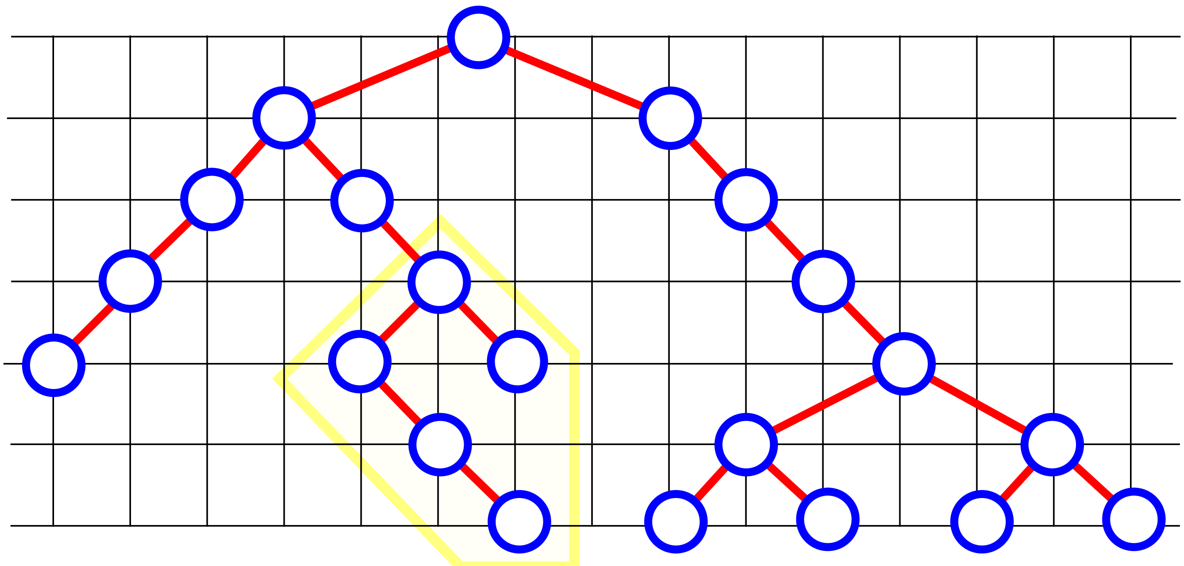


Properties of Recursive Level Drawing Algorithm for Binary Trees

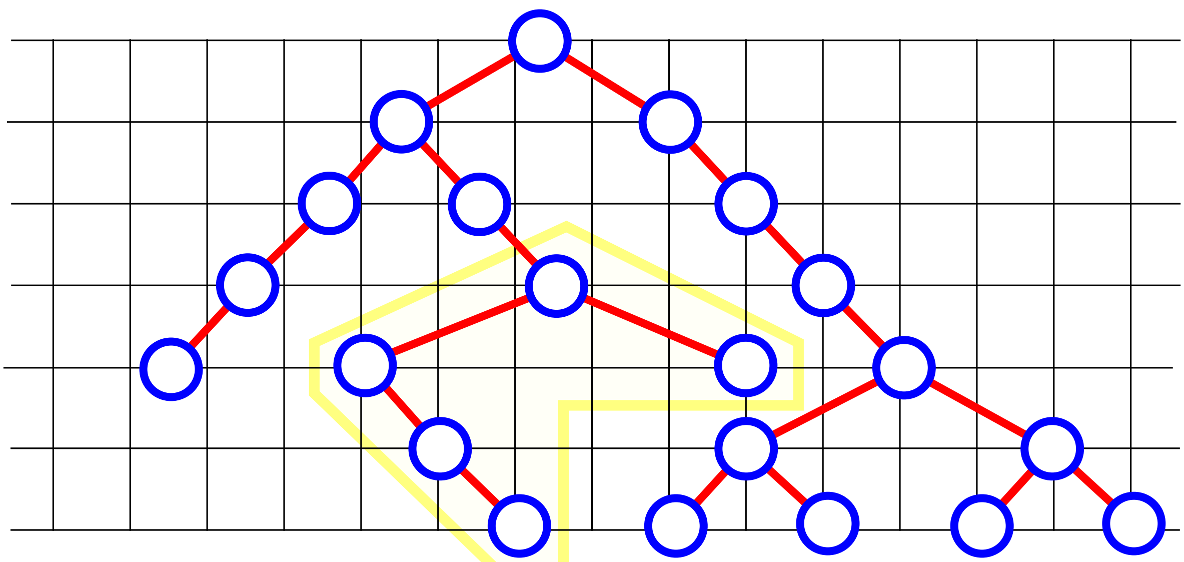


- ***centered*** level drawing
- "small" width
- display of symmetries and of isomorphic subtrees
- can be implemented to run in O(n) time
- can be extended to draw general rooted trees (e.g., root is placed at the average x-coordinate of its children)

Non Optimality of Recursive Tree Drawing Algorithm



drawing constructed by the algorithm

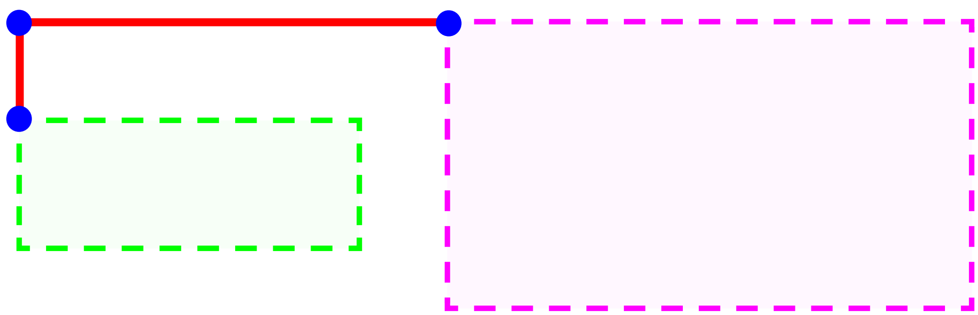


minimum width drawing

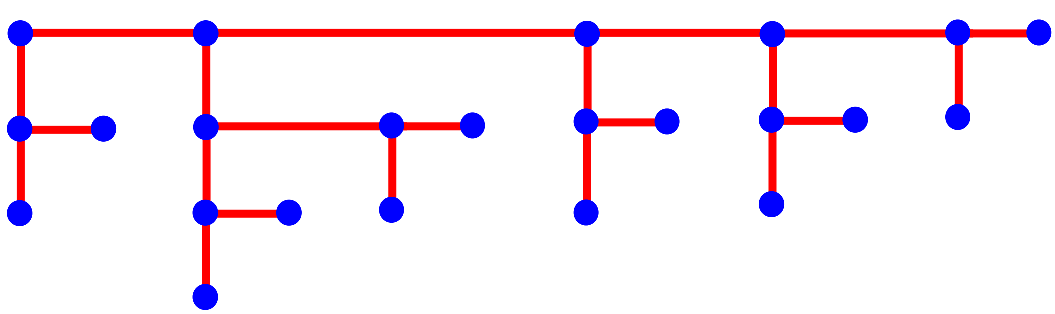
- minimizing the width is NP-hard if integer coordinates are required

Area-Efficient Drawings of Trees

- planar straight-line orthogonal upward grid drawing of a binary tree with ***O(n* log *n) area***, ***O(n) width***, and ***O(*log *n) height***
  [Crescenzi Di Battista Piperno 92]
  [Shiloach 76]
- draw the ***largest subtree*** "to the right" and the ***smallest subtree*** "below"

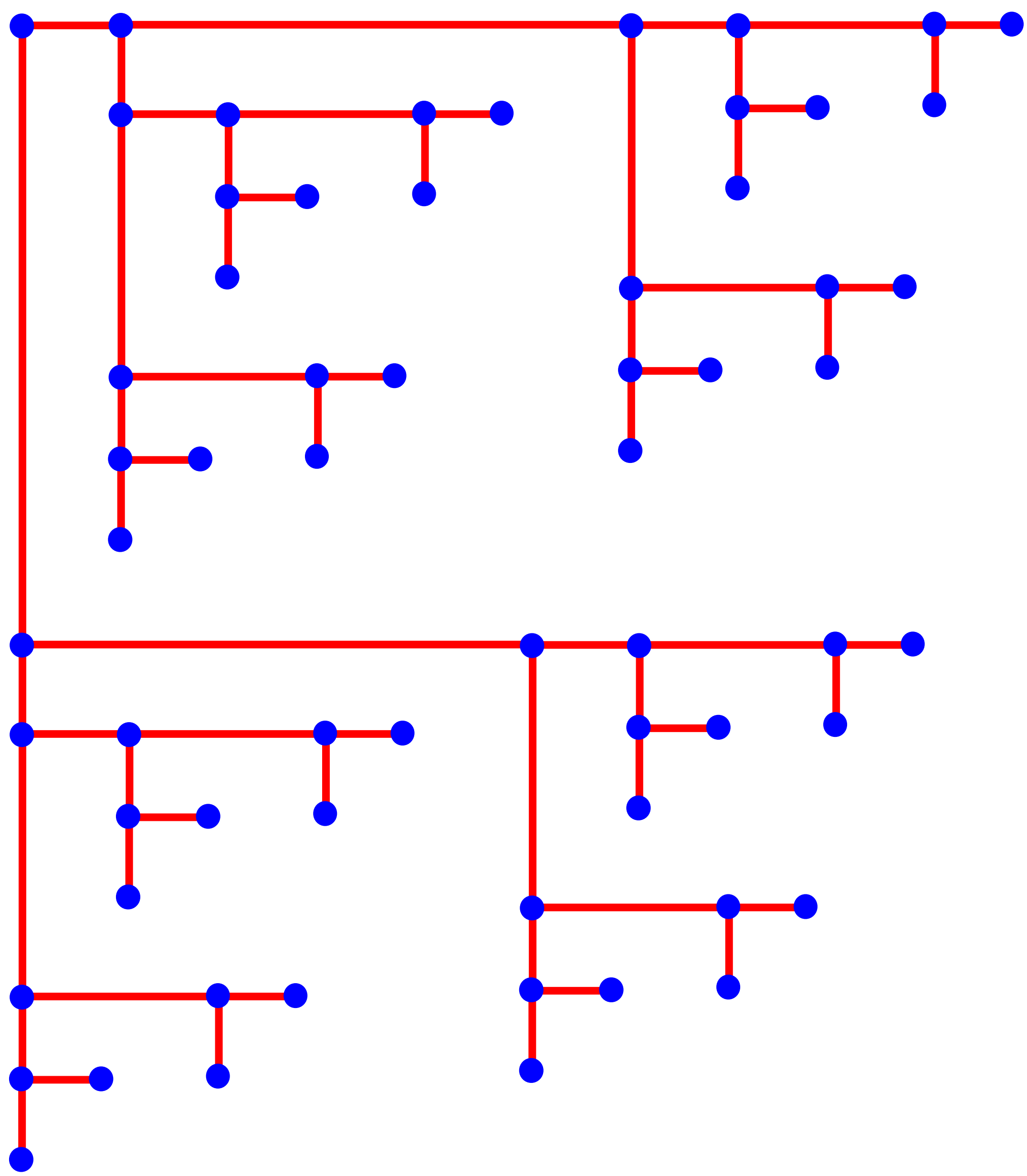


- Example:



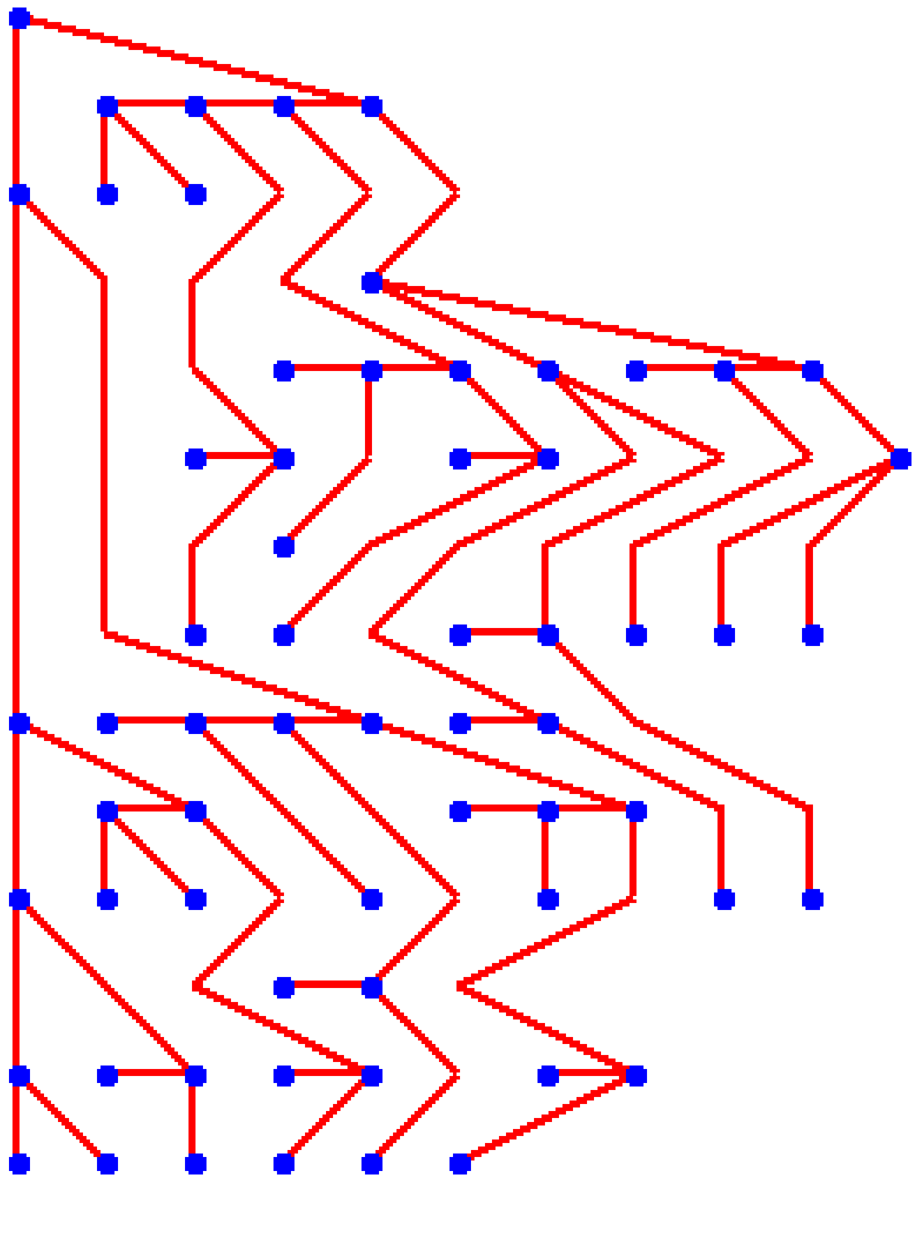
Area-Efficient Drawings of Trees

- planar straight-line upward grid drawings of ***AVL trees*** with ***O(n) area***
  [Crescenzi Di Battista Piperno 92]
  [Crescenzi Penna Piperno 95]



Area-Efficient Drawings of Trees

- planar polyline upward grid drawings with ***O(n) area***
  [Garg Goodrich Tamassia 93]



Area Requirement of Planar Drawings of Trees

upward level	$\Theta(n^2)$ [RT 83]
upward polyline	$\Theta(n)$ [GGT 93]
upward straight-line	$\Omega(n)$ $O(n \log n)$ [CDP 92]
upward orthogonal	$\Theta(n \log \log n)$ [GGT 93]
non-upward orthogonal	$\Theta(n)$ [L80, V91]
non-upward leaves-on-hull orthogonal	$\Theta(n \log n)$ [BK 80]

- ***Open Problem***: determine the area requirement of planar upward straight-line drawings of trees

Size of Planar Drawings of Binary Trees

- the ***size*** of a drawing is the maximum of its ***height*** and ***width***
- known bounds on the size of ***planar*** drawings of binary trees:

upward, straight-line level	$O(n)$ [RT 83]
upward, polyline	$\Theta(n^{1/2})$ [GGT93]
upward, straight-line orthogonal, *AVL trees*	$\Theta(n^{1/2})$ [CGKT96]
upward, straight-line orthogonal	$\Theta((n \log n)^{1/2})$ [CGKT96]

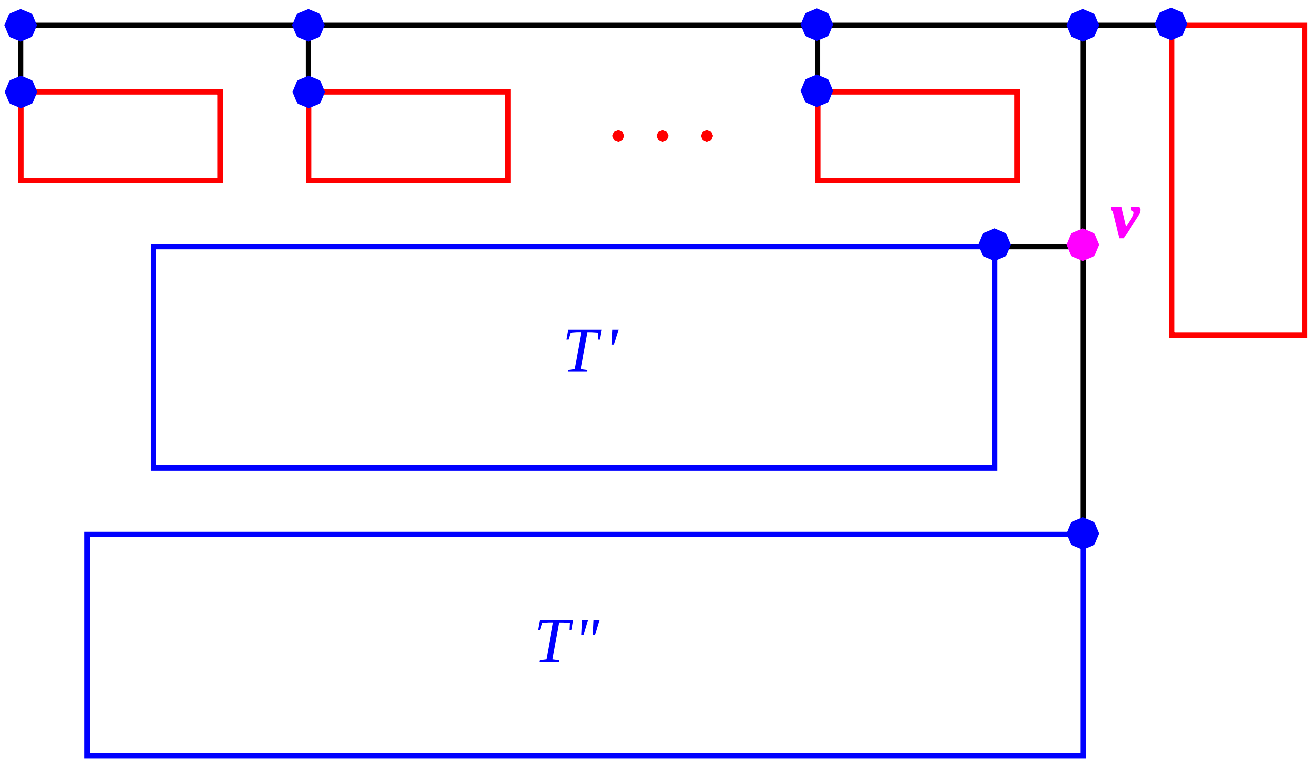
- ***Open Problem***: can $\Theta(n^{1/2})$ size be achieved for (nonupward) planar straight-line drawings of binary trees?

Planar Upward Straight-Line Drawings of Binary Trees with Optimal Size

- ***recursive winding*** technique [CGKT96]:
  - let N be number of nodes in the tree, and N(v) be the number of nodes in the subtree rooted at v
  - for each node u, swap children to have N(left(u)) ≤ N(right(u))
  - find the first node $\boldsymbol{v}$ on the rightmost path such that:

    $$N(right(\boldsymbol{v})) \leq N - (N \log N)^{1/2} < N(\boldsymbol{v})$$
  - draw the left subtrees on the path from the root to $\boldsymbol{v}$ with linear width (height) and logarithmic height (width)
  - draw recursively the subtrees T' and T'' of $\boldsymbol{v}$

Recursive Winding Drawing

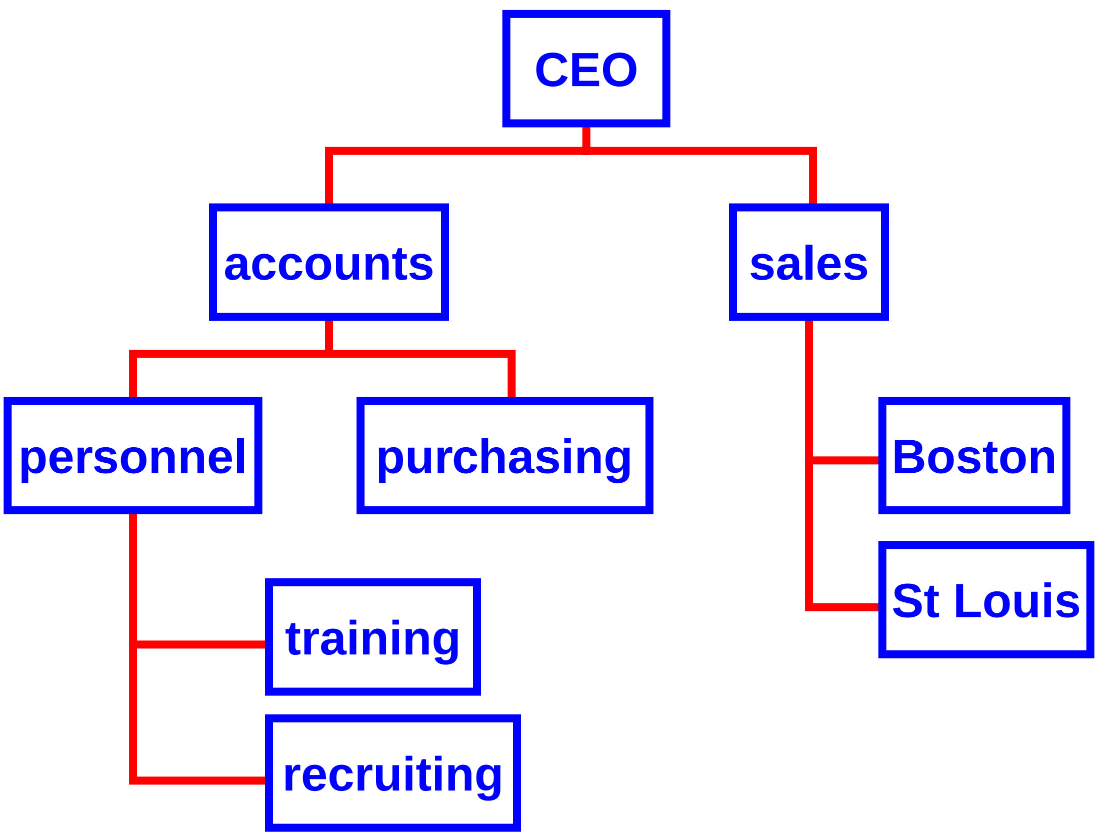


- recurrence relations for the width W(N) and height H(N):
  - W(N) max{W(N'), W(N''), A} + O(log N)
  - H(N) max{H(N') + H(N'') + O(log N), A}

  where:
  - A = $(N \log N)^{1/2}$
  - max(N', N'') ≤ N – A
- solution:
  - W(N)=H(N)= $O(N \log N)^{1/2}$

Tip-Over Drawings of Rooted Trees

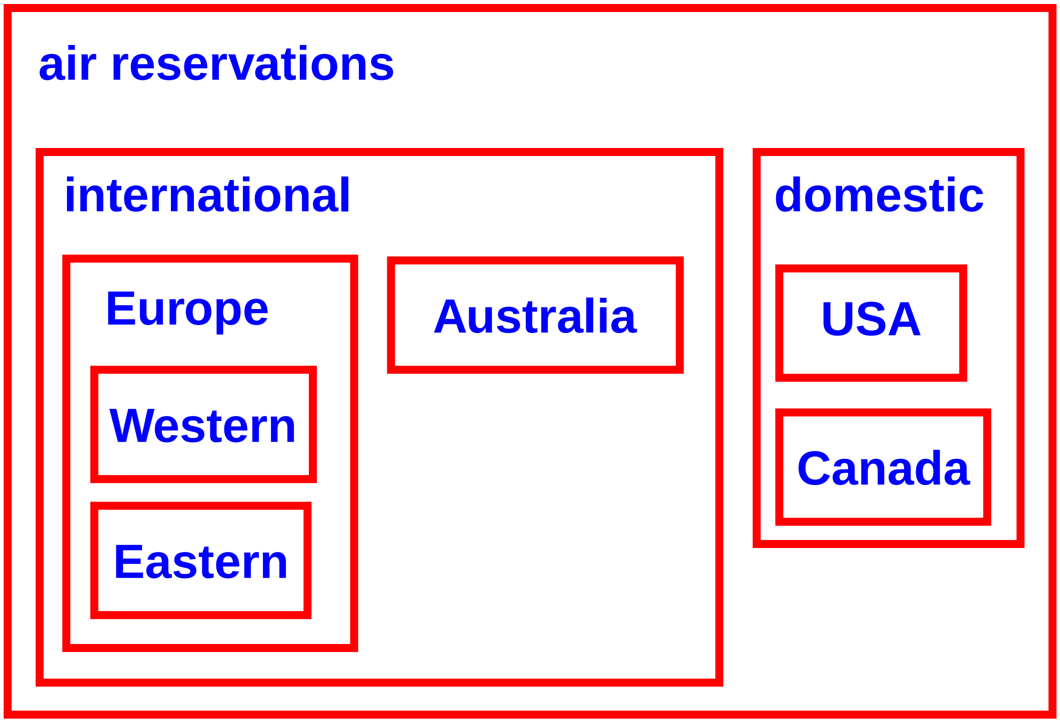
- ***Tip-over drawings*** are upward planar orthogonal drawings such that the children of a node:
  - are arranged either horizontally or vertically
  - share portions of the edges to the parent.



- Widely used in organization charts.
- Allow to better fit the drawing in a prescribed region.

Inclusion Drawings of Rooted Trees

- *Inclusion drawings* display the parent-child relationship by the inclusion between isothetic rectangles.



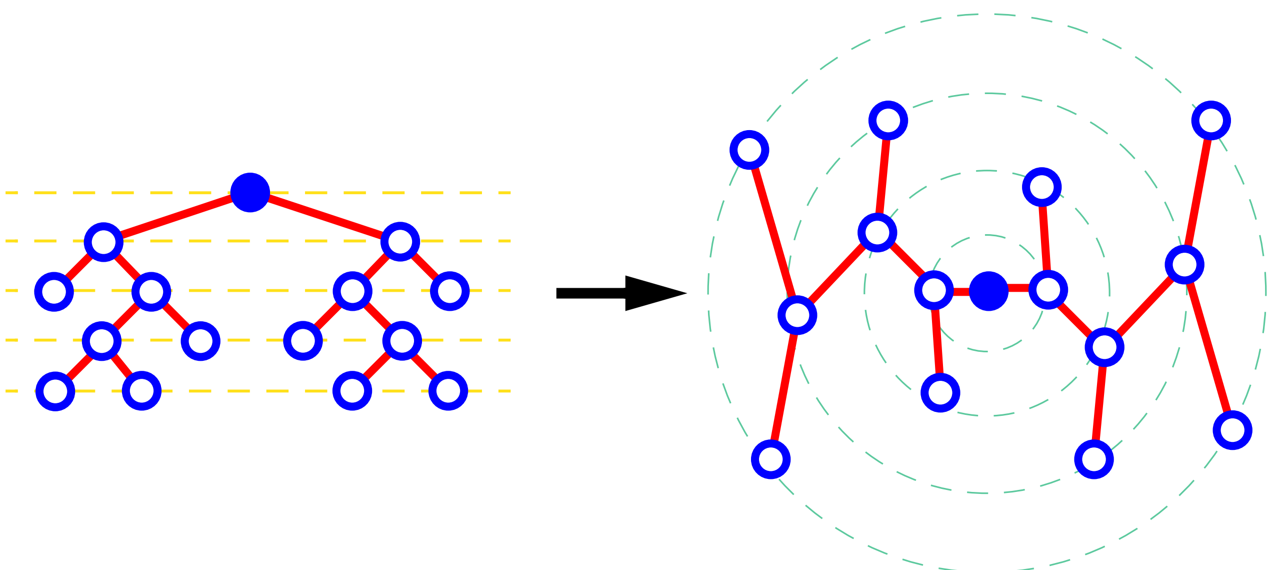
- Closely related to tip-over drawings.
- Used for displaying compound graphs (e.g., the union of a graph and a tree)
- Allow to better fit the drawing in a prescribed region

Area of Tip-Over and Inclusion Drawings

- Eades, Lin and Lin (1992) study of the area requirement of tip-over and inclusion drawings of rooted trees.
- The dimensions of the node labels are given as part of the input.
- ***Minimizing the area*** of the drawing is:
  - ***NP-hard for general trees***
  - computable in ***polynomial time*** for ***balanced trees*** with a ***dynamic programming*** algorithm
- Similar results for the following problems:
  - minimizing the ***perimeter*** of the drawing.
  - minimizing the ***width*** for a given height
  - minimizing the ***height*** for a given width

How to Draw Free Trees

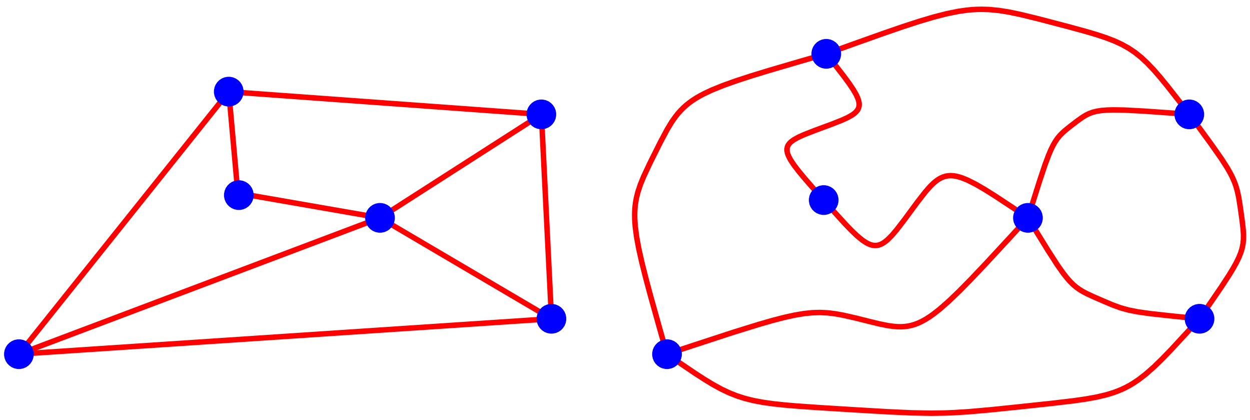
- ***Free trees*** are connected graphs without cycles and do not represent hierarchical relationships (e.g., spanning trees)
- Level drawings of rooted trees yield ***radial drawings*** of free trees:
  - root the free tree T at its ***center*** (node with minmax distance from the leaves), which gives a rooted tree T'
  - construct a level drawing Δ' of T'
  - use a geometric transformation (***cartesian*** → ***polar***) to obtain from Δ' a radial drawing Δ of T



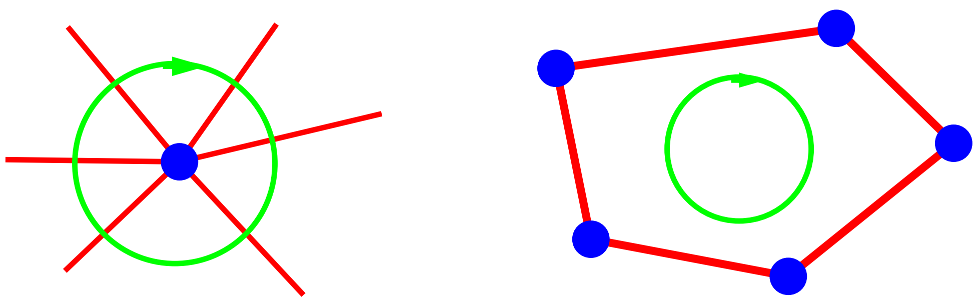
Planar Undirected Graphs

Planar Drawings and Embeddings

- a ***planar embedding*** is a class of topologically equivalent planar drawings



- a planar embedding prescribes
  - the ***star*** of edges around each vertex
  - the ***circuit*** bounding each face



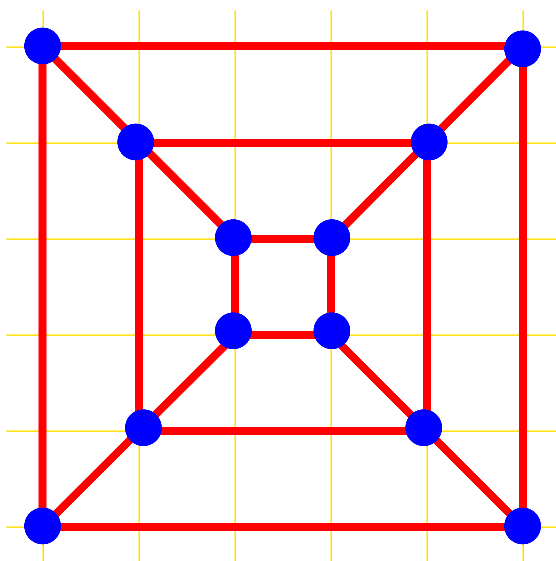
- the number of distinct embeddings is exponential in the worst case
- triconnected planar graphs have a unique embedding

The Complexity of Planarity Testing

- Planarity testing and constructing a planar embedding can be done in ***linear time***:
  - ***depth-first-search***
    [Hopcroft Tarjan 74]
    [de Fraysseix Rosenstiehl 82]
  - ***st-numbering and PQ-trees***
    [Lempel Even Cederbaum 67]
    [Even Tarjan 76]
    [Booth Lueker 76]
    [Chiba Nishizeki Ozawa 85]
- The above methods are ***complicated*** to understand and implement
- ***Open Problem:***
  - devise a ***simple*** and ***efficient*** planarity testing algorithm.

Planar Straight-Line Drawings

- [Hopcroft Tarjan 74]: planarity testing and constructing a planar embedding can be done in O(n) time
- [Fary 48, Stein 51, Steinitz 34, Wagner 36]: every planar graph admits a planar straight-line drawing

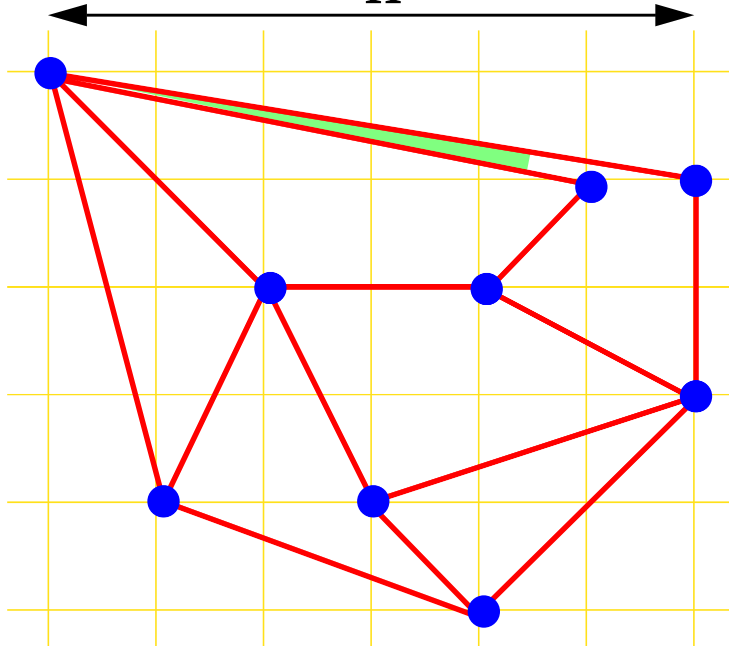


- Planar straight-line drawings may need $\Omega(n^2)$ area
- [de Fraysseix Pach Pollack 88, Schnyder 89, Kant 92]: $O(n^2)$-area planar straight-line grid drawings can be constructed in O(n) time

Planar Straight-Line Drawings: Angular Resolution

- $O(n^2)$-area drawings may have $\rho = O(1/n^2)$

n



1

- [Garg Tamassia 94]:
  - ***Upper bound*** on the angular resolution:

$$\rho = O\left(\sqrt{\frac{\log d}{d^3}}\right)$$

  - ***Trade-off*** (area vs. angular resolution):

$$A = \Omega(c^{\rho n})$$

- [Kant 92] Computing the optimal angular resolution is ***NP-hard***.

Planar Straight-Line Drawings: Angular Resolution

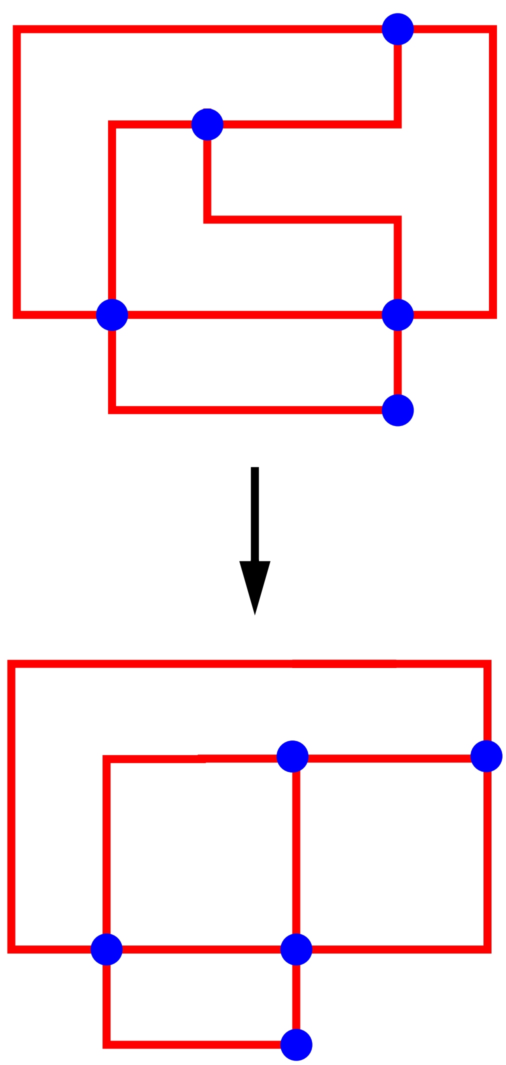
- [Malitz Papakostas 92]: the angular resolution depends on the degree only:

$$\rho = \Omega\left(\frac{1}{7^d}\right)$$

- Good angular resolution can be achieved for special classes of planar graphs:
  - ***outerplanar graphs***, $\rho = O(1/d)$ [Malitz Papakostas 92]
  - ***series-parallel graphs***, $\rho = O(1/d^2)$ [Garg Tamassia 94]
  - ***nested-star graphs***, $\rho = O(1/d^2)$ [Garg Tamassia 94]
- ***Open Problems***:
  - can we achieve $\rho = O(1/d^k)$ (k a small constant) for all planar graphs?
  - can we efficiently compute an ***approximation*** of the optimal angular resolution?

Planar Orthogonal Drawings: Minimization of Bends

- given planar graph of degree ≤ 4, we want to find a planar orthogonal drawing of G with the minimum number of bends

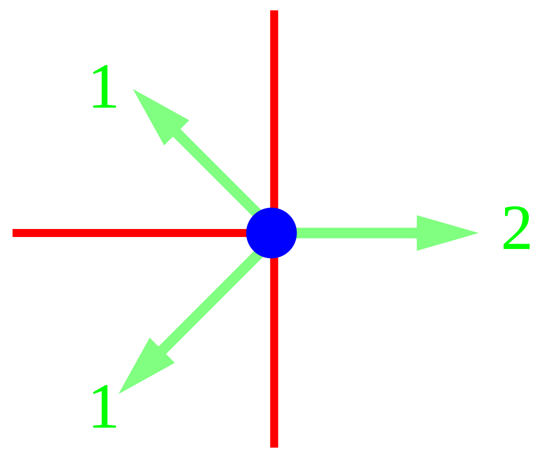


Minimization of Bends in Planar Orthogonal Drawings

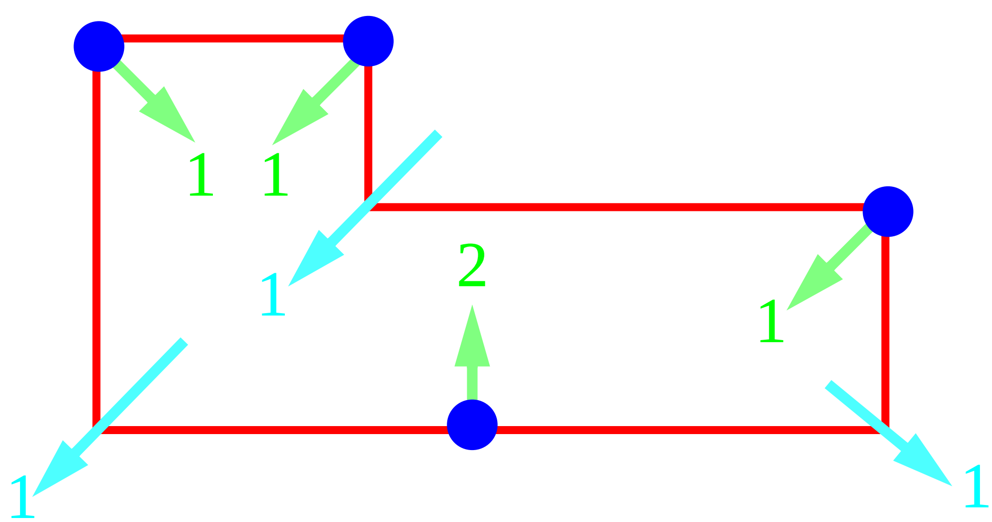
- [Tamassia 87]
  - $O(n^2 \log n)$-time bend minimization for fixed embedding
- [Di Battista Liotta Vargiu 93]
  - polynomial-time bend minimization for degree-3 and series-parallel graphs
- [Tamassia Tollis 89]
  - O(n)-time approximation with O(n) bends
- [Garg Tamassia 93]
  - minimization of bends is NP-hard
  - approximation with $O(opt + n^{1-\varepsilon})$ bends is NP-hard
  - rectilinear planarity testing is NP-complete

Network Flow Model

- a unit of flow is a 90° angle
- a vertex (source) produces 4 units



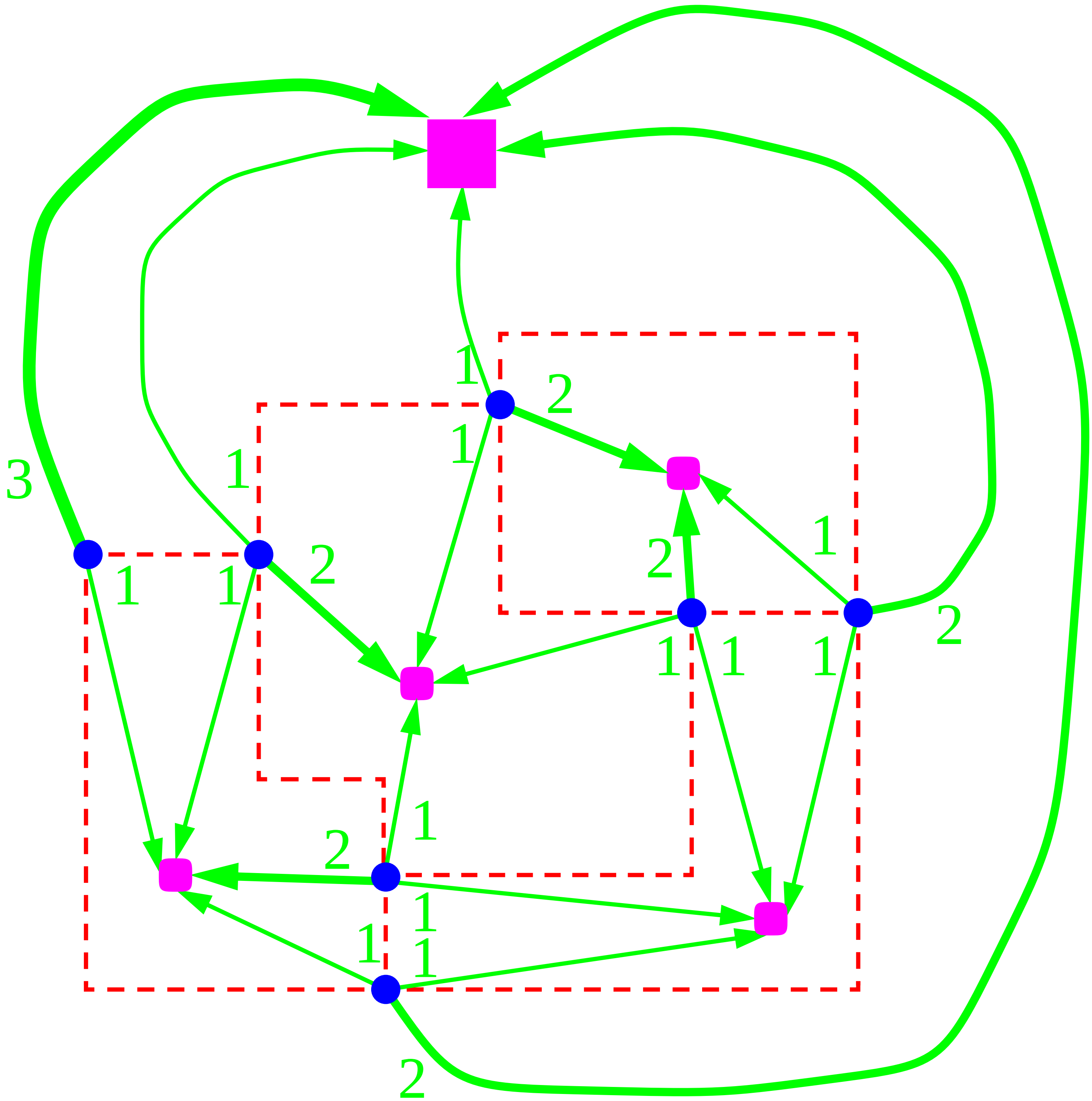
- a face f (sink) consumes 2 deg(f) – 4 units (deg(f) + 4 for the external face)



- Edges transport flow across faces

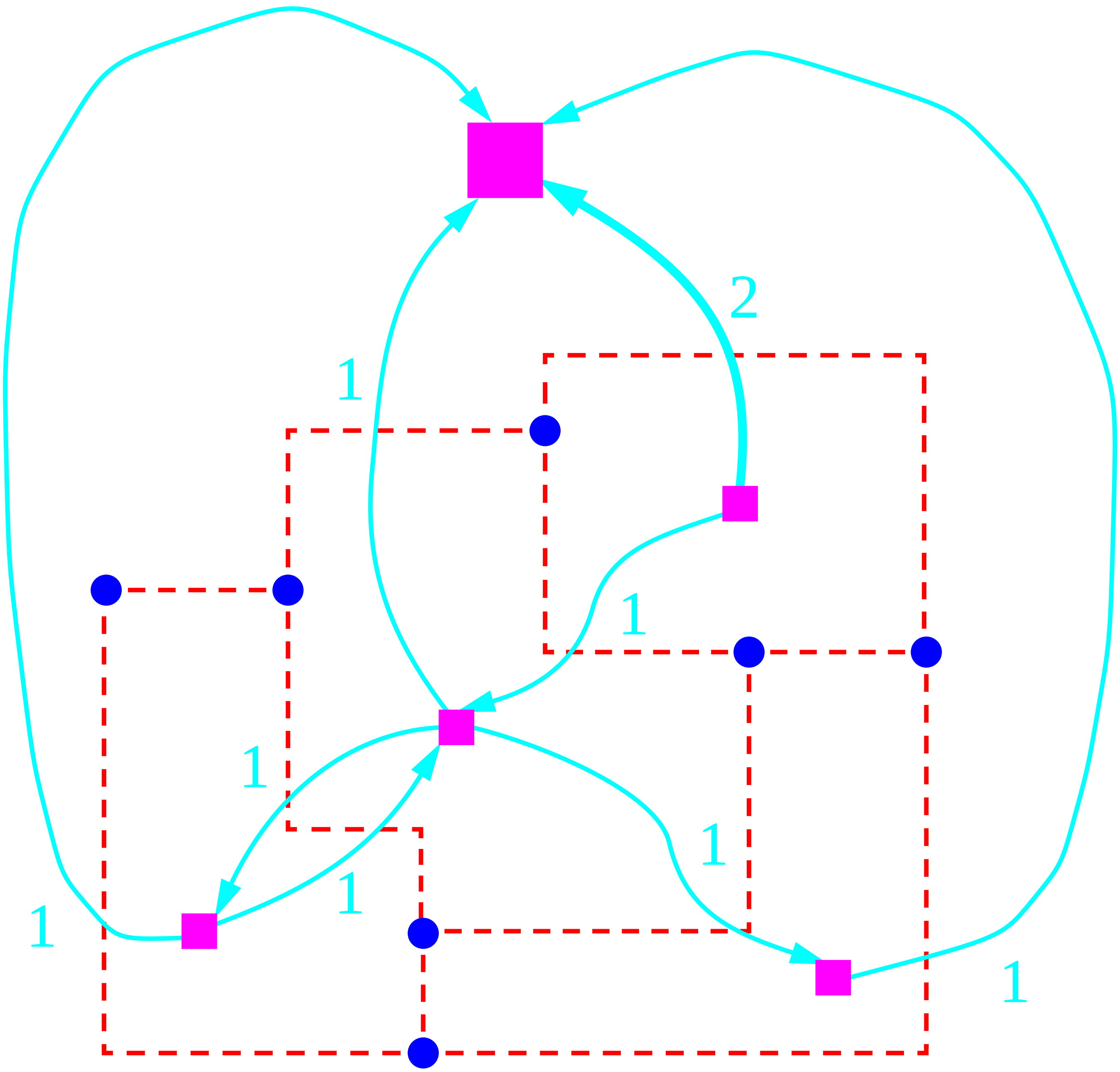
Flow Network

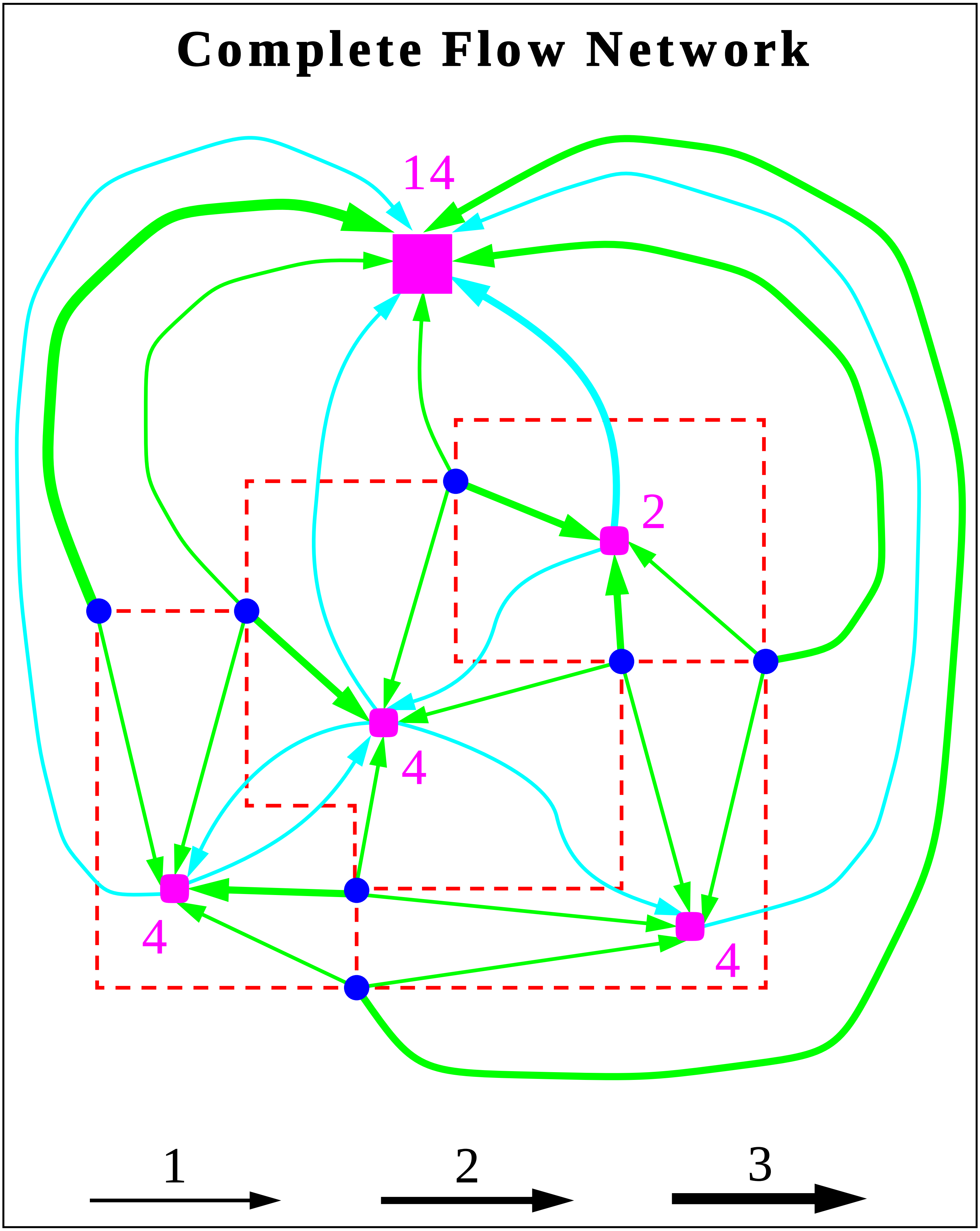
- vertex-face arcs: flow ≥ 1, cost = 0



Flow Network

- face-face arcs: flow ≥ 0, cost = 1




Complete Flow Network
14
2
4
4
4
1
2
3

Correctness of Flow Model

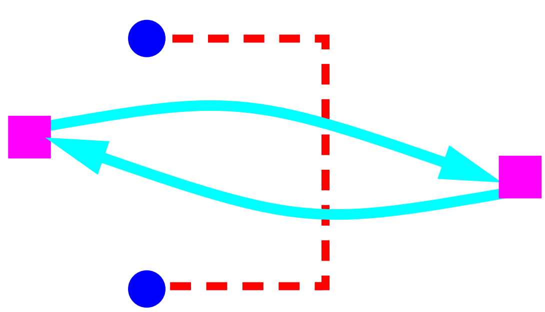
- supply of sources = demand of sinks $\leftrightarrow$ Euler's formula
- flow conservation at vertex $\leftrightarrow$ $\sum$ angles around vertex = 360°
- flow conservation at face $\leftrightarrow$ (# 90° angles) – (# 270° angles) = 4
- cost of flow $\leftrightarrow$ # bends
- flow in N $\leftrightarrow$ drawing of G
- minimum cost flow $\leftrightarrow$ optimal drawing

Theorem [Tamassia 87] Computing the minimum number of bends for an embedded graph G is equivalent to computing a minimum cost flow in network N, and takes $O(n^2 \log n)$ time

Open Problem: reduce the time complexity of bend minimization.

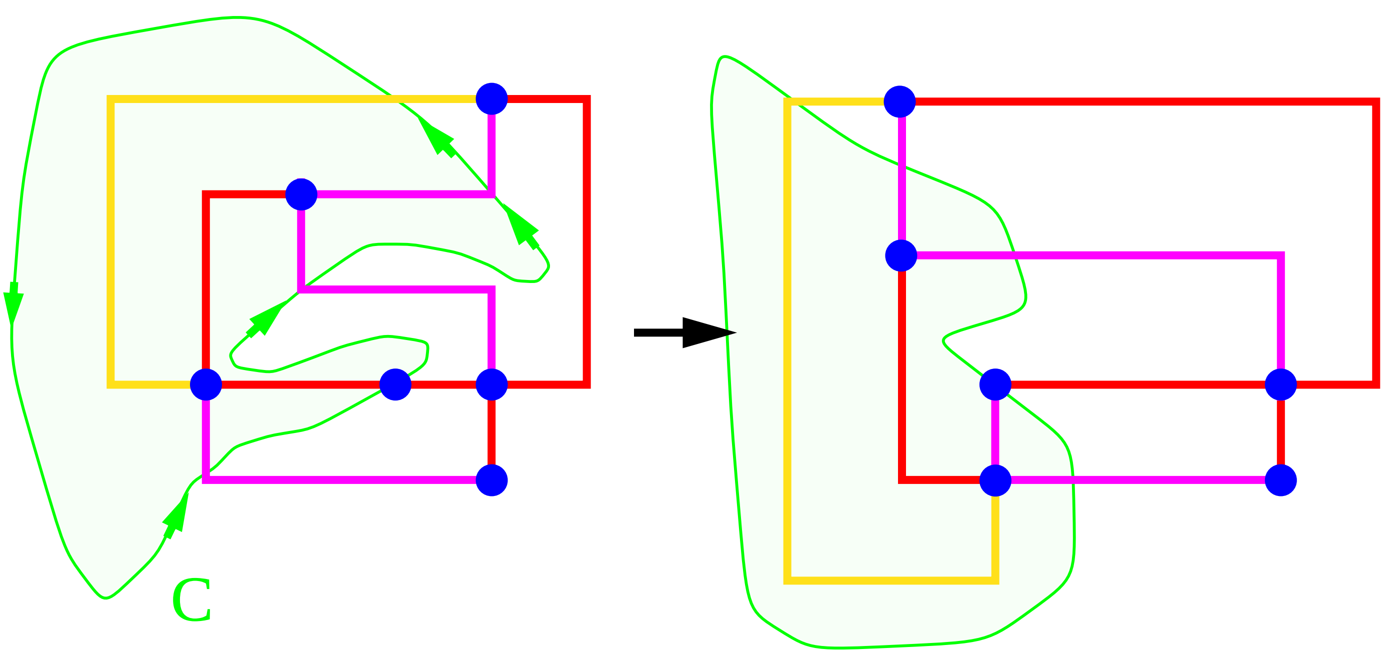
Constrained Bend Minimization

- the network flow model allows us to minimize bends subject to ***shape constraints***
  - prescribed angles around a vertex
  - prescribed bends along an edge
  - upper bound on the number of bends on an edge
- the above ***shape constraints*** on the drawing can be expressed by setting appropriate ***capacity constraints*** on the edges of the network
- E.g., we can prescribe a maximum of 2 bends on a given edge ***e*** by setting equal to 2 the capacity of the ***face-face arcs*** associated with ***e***



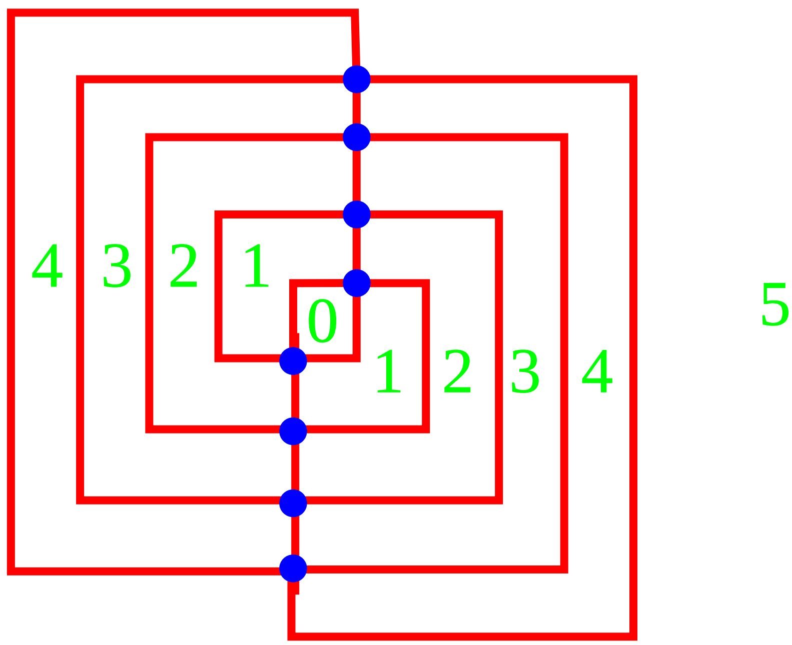
Characterization of Bend-Minimal Drawings

- A drawing has the minimum number of bends if and only if there is no oriented closed curve C such that
  - vertices are intersected by C entering from angles ≥ 180°
  - (# edges crossed by C from 90° or 180°) < (# edges crossed by C from 270°)
- If such a curve exists, "rotating" the portion of the drawing inside C reduces the number of bends



Proving the Optimality of a Drawing

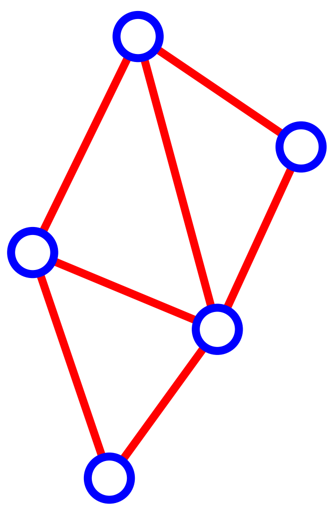
- potential Φ on each face

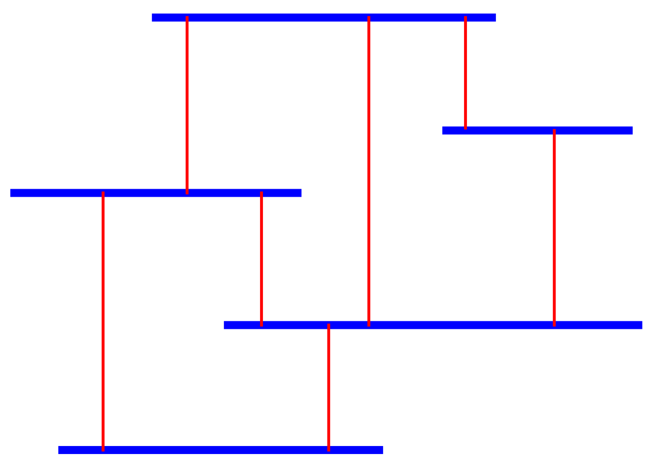


- vertices cannot be traversed by C
- C traverses edge from 270° $\Rightarrow$ $\Delta\Phi_i = -1$
- C traverses edge from 90° $\Rightarrow$ $\Delta\Phi_i = +1$
- bends removed going "inward" and inserted going "outward" $\Delta B_i + \Delta\Phi_i = 0$
- C is a closed curve $\Rightarrow$ $\sum_i \Delta\Phi_i = 0$
- Hence, $\sum_i \Delta B_i = 0$

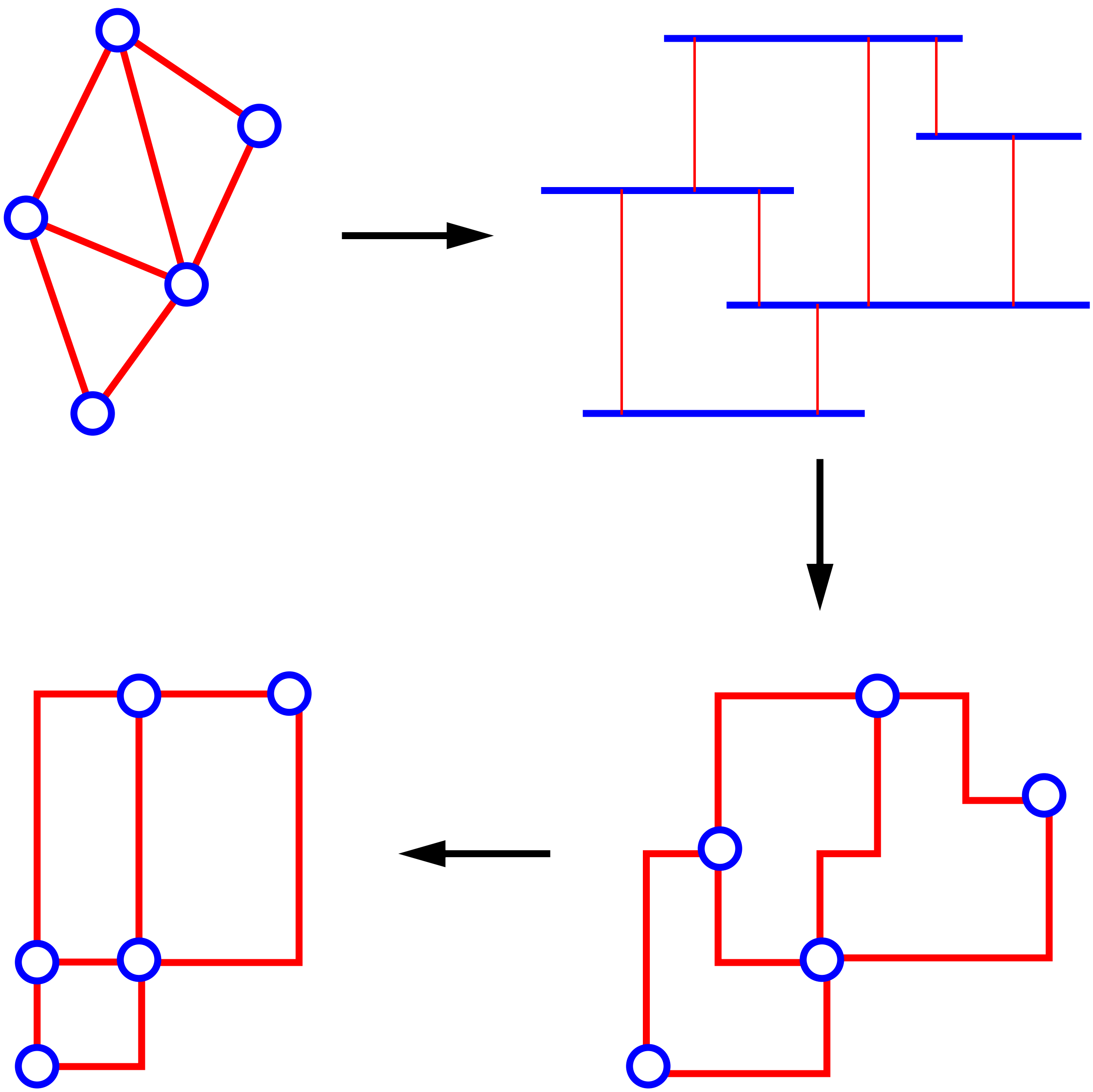
Visibility Representation

- vertices → horizontal segments
- edges → vertical segments
- can be constructed in O(n) time
- preliminary step for drawing algorithms





From Visibility Representations to Orthogonal Drawings



Heuristic Algorithm for Bend Minimization

1. Construct visibility representation
2. Transform visibility representation into a preliminary drawing
3. Apply bend-stretching transformations
4. Compact orthogonal representation

Runs in O(n) time and can be parallelized

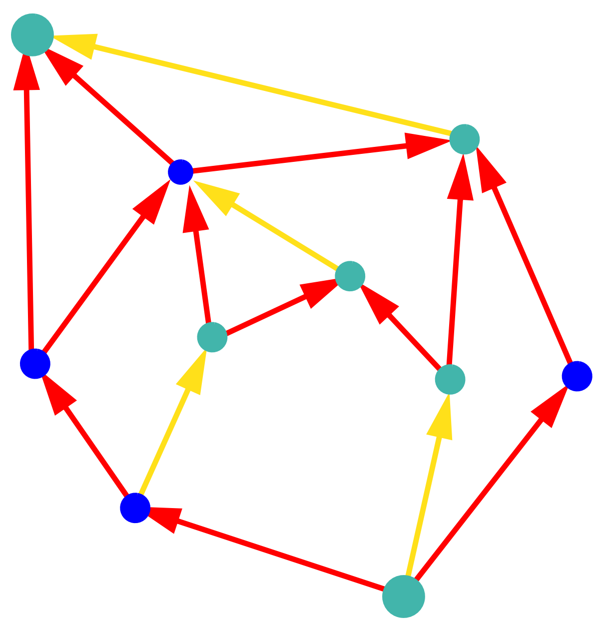
At most 2n + 4 bends if G is biconnected (2.4n + 2 otherwise)

$O(n^2)$ area

Planar Directed Graphs

Upward Planarity Testing

- upward planarity testing for ordered sets has the same complexity as for general digraphs (insert dummy vertices on transitive edges)
- [Kelly 87, Di Battista Tamassia 87]: upward planarity is equivalent to subgraph inclusion in a planar st-digraph (planar acyclic digraph with one source and one sink, both on the external face)



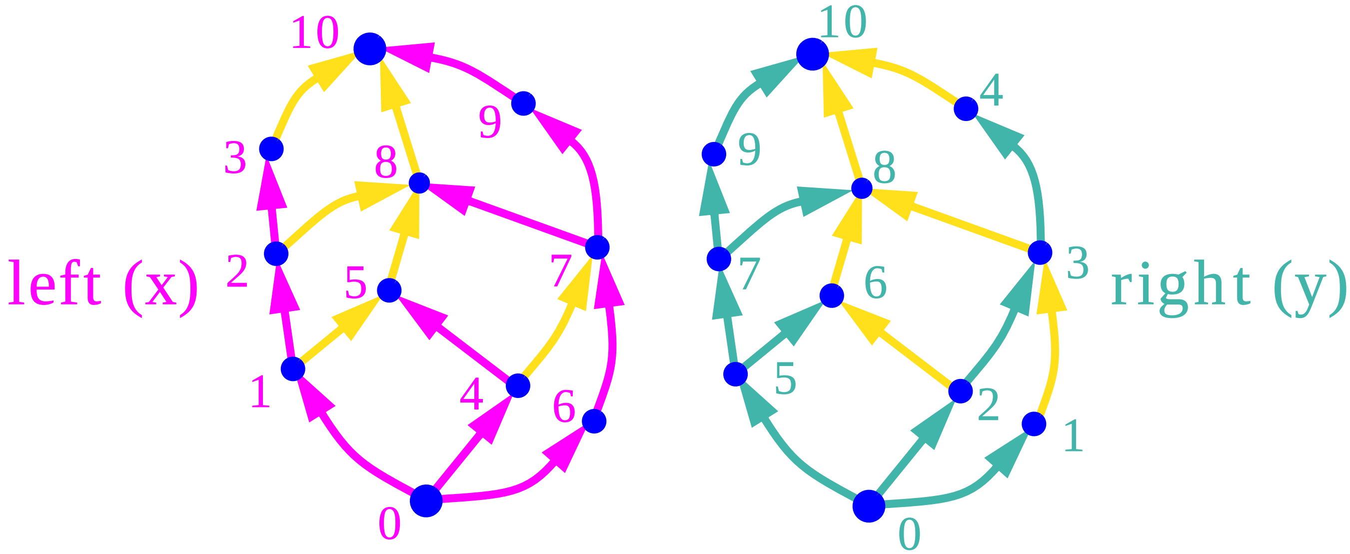
- [Kelly 87, Di Battista Tamassia 87]: upward planarity is equivalent to upward straight-line planarity

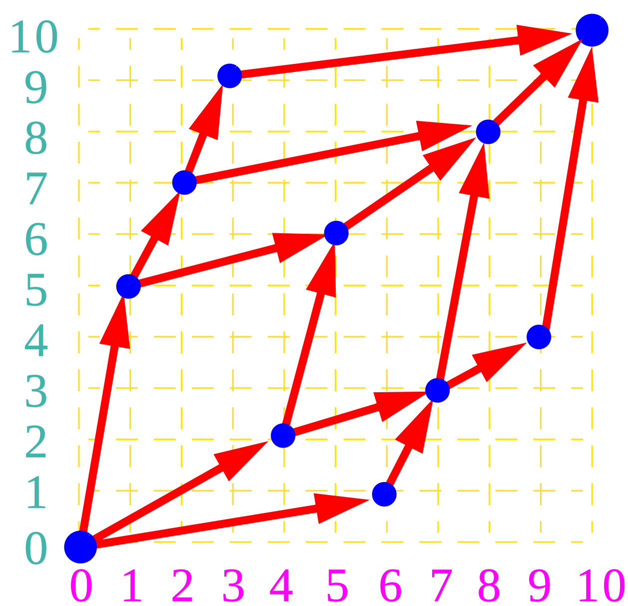
Complexity of Upward Planarity Testing

- [Bertolazzi Di Battista Liotta Mannino 91]
  - $O(n^2)$-time for fixed embedding
- [Hutton Lubiw 91]
  - $O(n^2)$-time for single-source digraphs
- [Bertolazzi Di Battista Mannino Tamassia 93]
  - $O(n)$-time for single-source digraphs
- [Garg Tamassia 93]
  - NP-complete

How to Construct Upward Planar Drawings

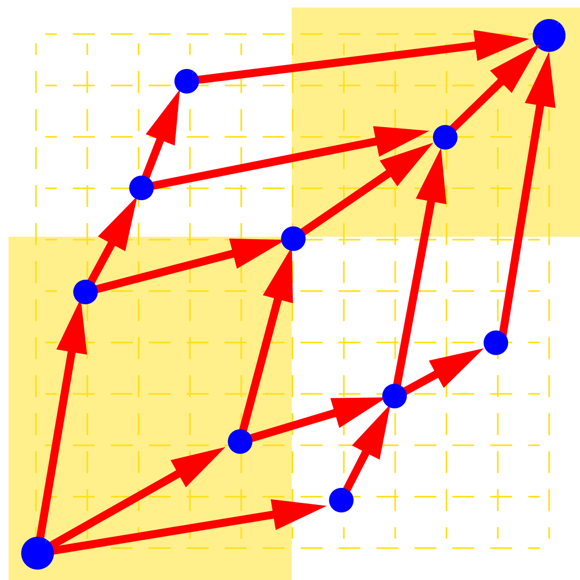
- Since an upward planar digraph is a subgraph of a ***planar st-digraph***, we only need to know how to draw planar st-digraphs
- If G is a planar st-digraph without transitive edges, we can use the ***left***/***right*** numbering method to obtain a ***dominance drawing***:



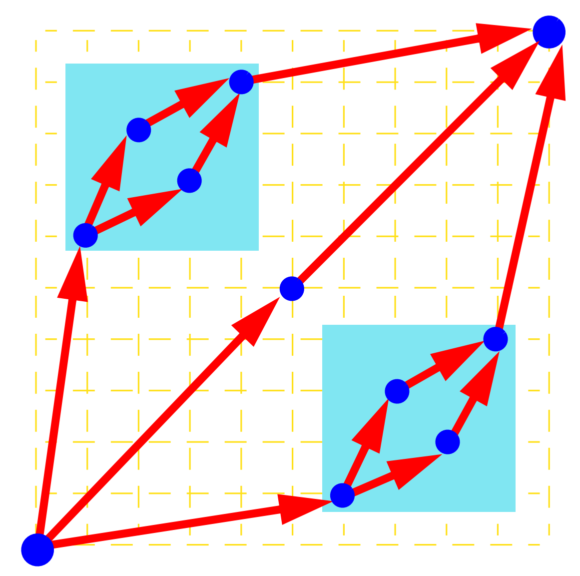


Properties of Dominance Drawings

- ***Upward, planar, straight-line, $O(n^2)$ area***
- The ***transitive closure*** is visualized by the geometric dominance relation

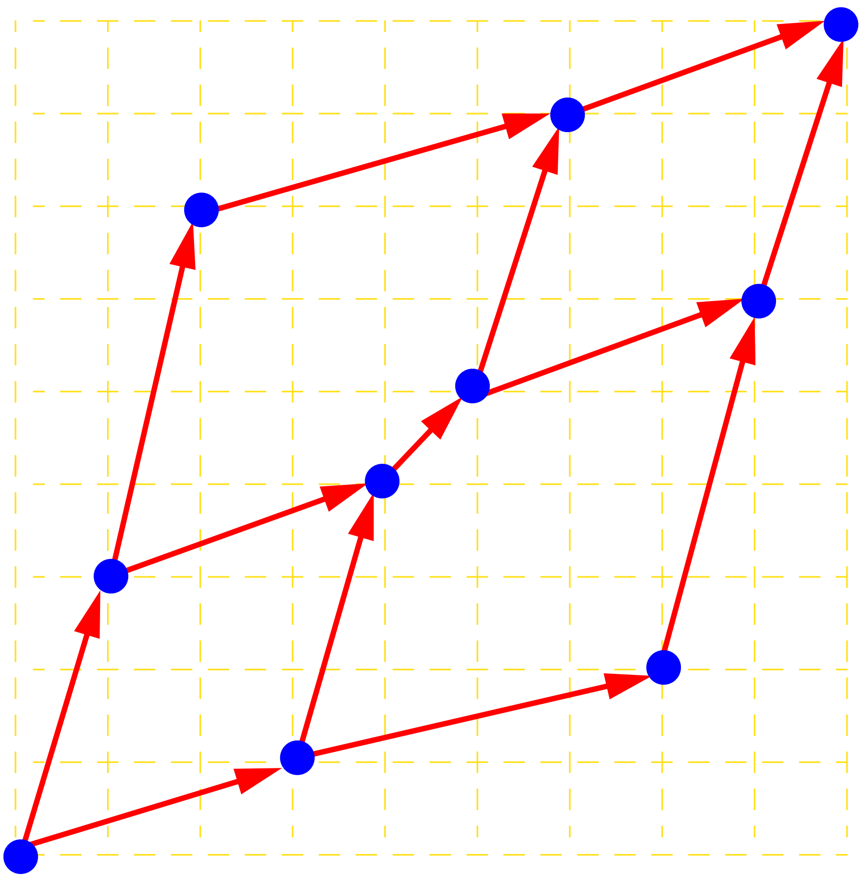


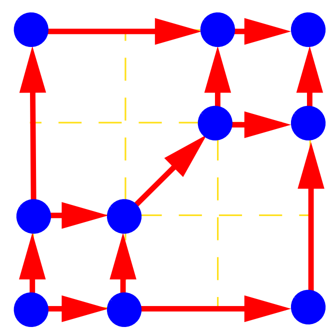
- ***Symmetries*** and ***isomorphisms*** of ***st-components*** are displayed



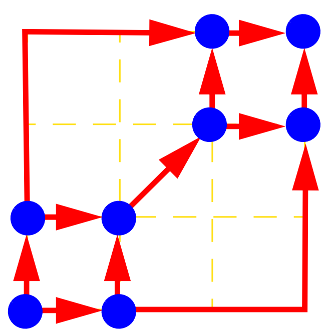
More on Dominance Drawings

- A variation of the left/right numbering yields dominance drawings with ***optimal area***



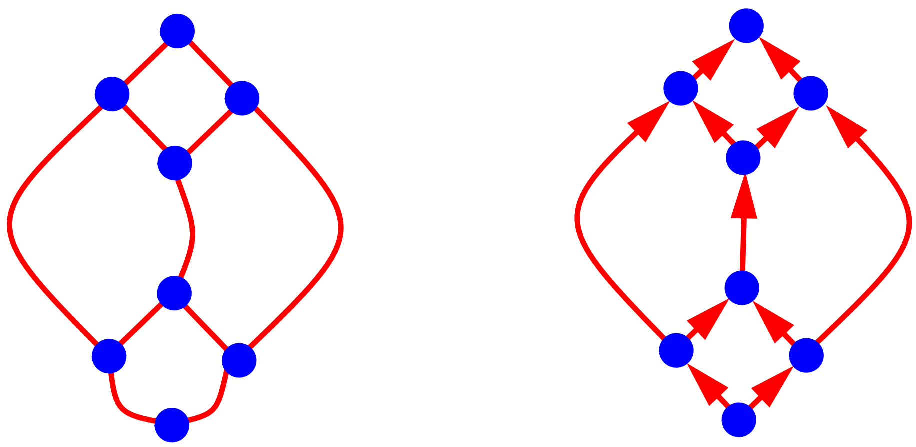


- Dummy vertices are inserted on transitive edges and are displayed as bends (upward planar polyline drawings)

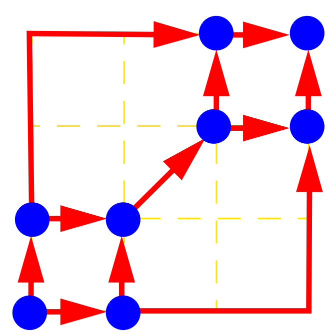


Planar Drawings of Graphs and Digraphs

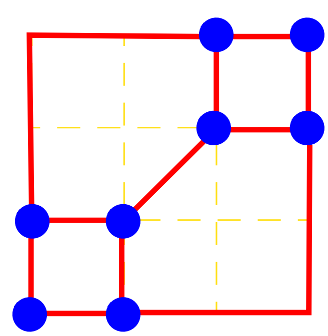
- We can use the techniques for dominance drawings also for undirected planar graphs:
  - orient G into a planar st-digraph G'



  - construct a dominance drawing of G'



  - erase arrows ...



General Undirected Graphs

Algorithmic Strategies for Drawing General Undirected Graphs

- ***Planarization method***
  - if the graph is nonplanar, ***make it planar***! (by placing dummy vertices at the crossings)
  - use one of the drawing algorithms for planar graphs

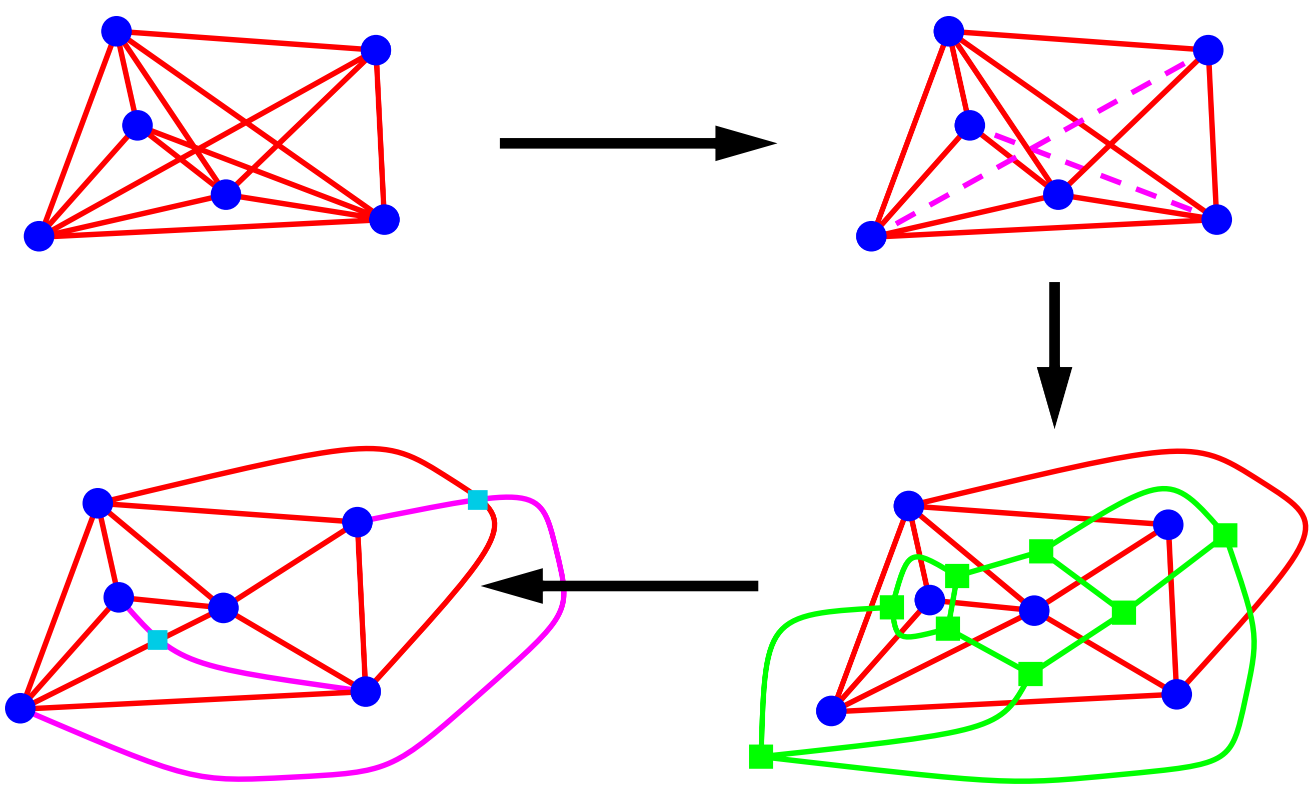
  e.g., GIOTTO [Tamassia Batini Di Battista 87]
- ***Orientation method***
  - ***orient*** the graph into a digraph
  - use one the drawing algorithms for digraphs
- ***Force-Directed method***
  - define a ***system of forces*** acting on the vertices and edges
  - find a ***minimum energy state*** (solve differential equations or simulate the evolution of the system)

  e.g., Spring Embedder [Eades 84]

A Simple Planarization Method

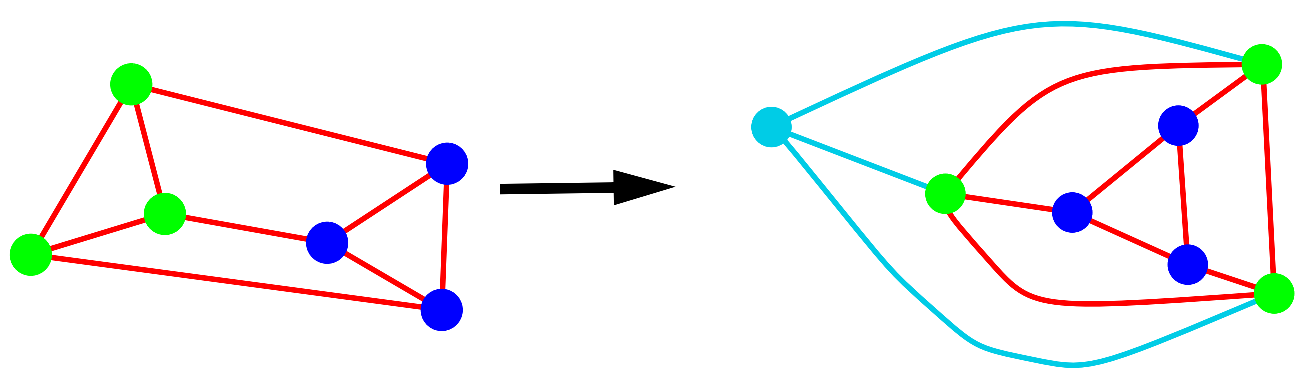
use an ***on-line planarity testing*** algorithm

1. try adding the edges one at a time, and divide them into "***planar***" (accepted) and "***nonplanar***" (rejected)
2. construct a planar embedding of the subgraph of the planar edges
3. add the nonplanar edges, one at a time, to the embedding, minimizing each time the number of ***crossings*** (shortest path in ***dual graph***)



Topological Constraints in the Planarization Method

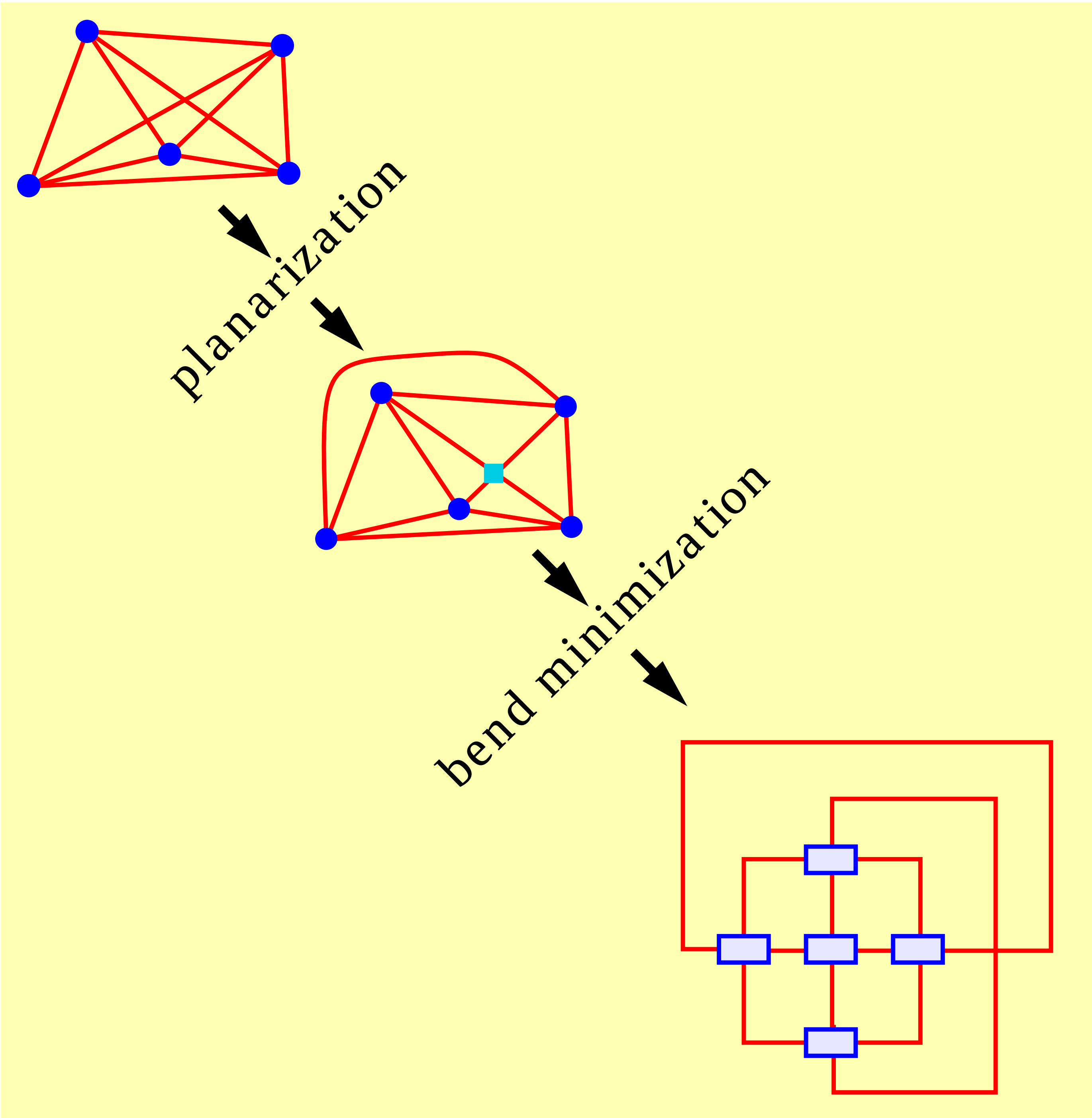
- a limited constraint satisfaction capability exists within the planarization methods
- ***Example:*** draw the graph such that the edges in a given set ***A*** have ***no crossings***
  - in Step 1, try adding first the edges in ***A***
  - in Step 3, put a large “crossing cost” on the planar edges in ***A***, and add first the nonplanar edges in ***A*** (if any)
- ***Example:*** draw the graph such the vertices of ***subset U*** are on the ***external boundary***
  - add a ***fictitious vertex v*** and edges from v to all the vertices in ***U***
  - let ***A*** be the set of edges (u,v), with u in ***U***
  - impose the above constraint



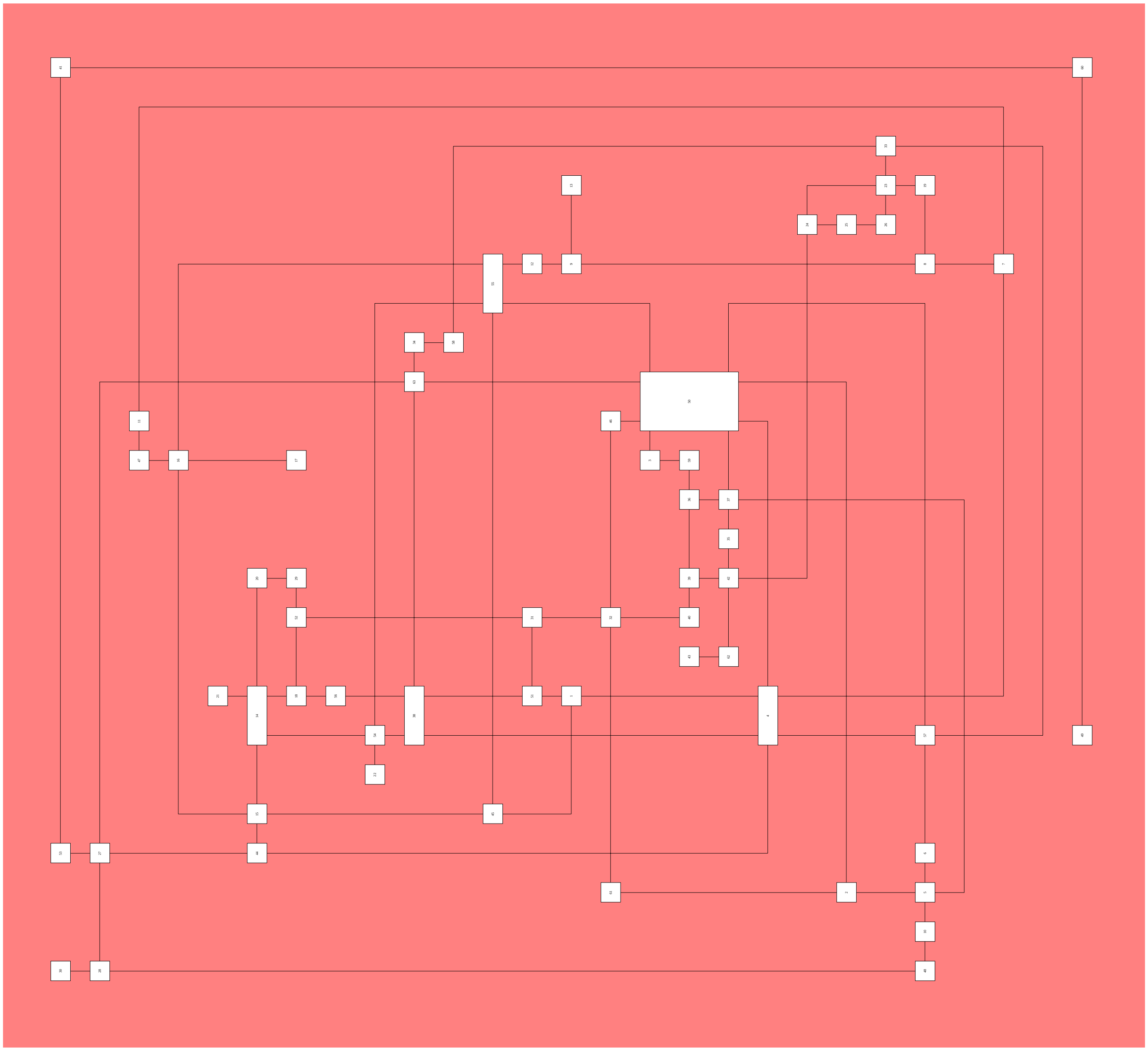
GIOTTO

[Tamassia Di Battista Batini 88]

- time complexity: $O((N+C)^2 \log N)$



Example



Constraint Satisfaction in GIOTTO

- ***topological constraints***
  - vertices on external face
  - edges without crossings
  - grouping of vertices
- ***shape constraints***
  - subgraphs with prescribed orthogonal shape
  - edges without bends
- topological contraints have ***priority*** over shape contraints because the algorithm assigns first the topology and then the orthogonal shape
- ***grouping is only topological***
- ***no position constraints***
- ***no length contraints***

Advantages and Disadvantages of Planarization Techniques

Pro:

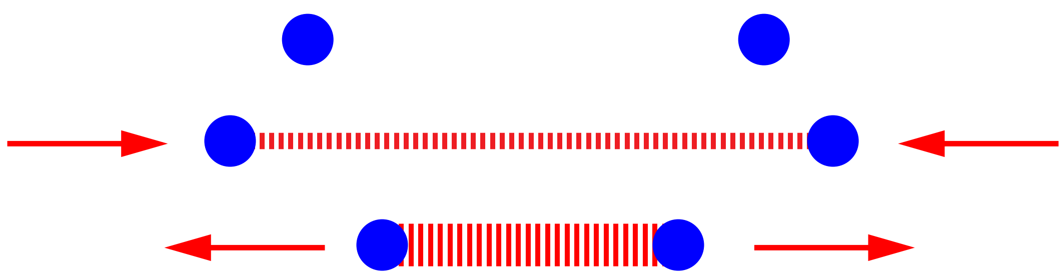
- ***fast*** running time
- ***applicable*** to straight-line, orthogonal and polyline drawings
- supported by ***theoretical results*** on planar drawings
- ***works well*** in practice, ***also for large graphs***
- limted ***constraint satisfaction*** capability

Con:

- relatively ***complex*** to implement
- ***topological transformations*** may alter the user's mental map
- ***difficult to extend to 3D***
- ***limted constraint satisfaction*** capability

The Spring Embedder
[Eades 1984]

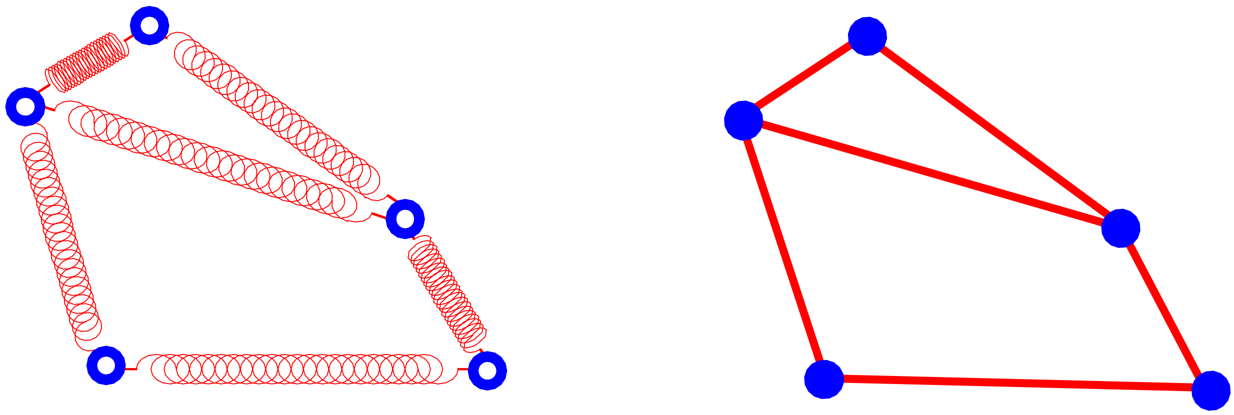
- replace the edges by ***springs*** with unit natural length
- connect nonadjacent vertices with additional springs with infinite natural length
- recall that the springs attract the endpoints when stretched, and repel the endpoints when compressed



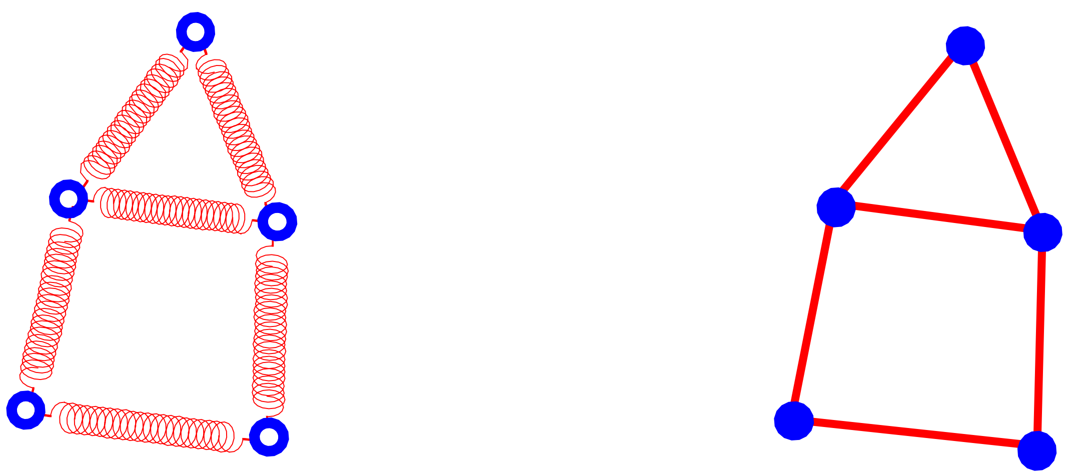
- start with an initial random placement of the vertices
- let the system go ... (assume there is ***friction*** so that a stable minimum energy state is eventually reached)

Example

- initial configuration



- final configuration

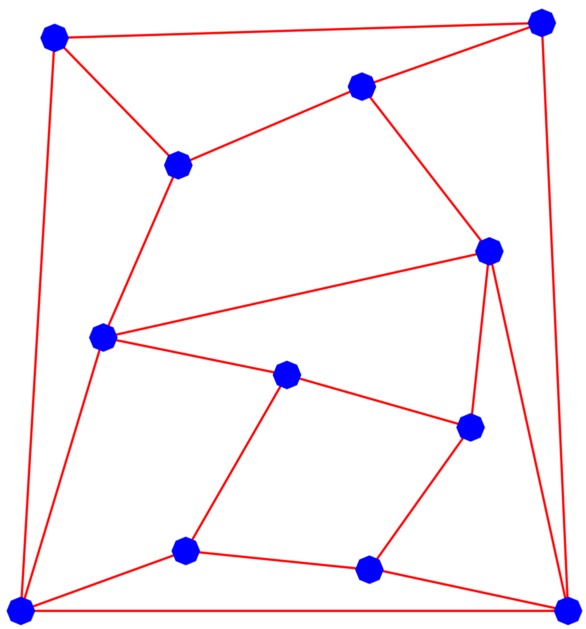


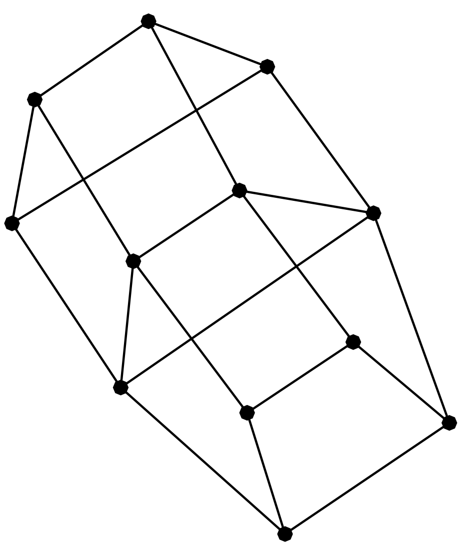
Other Force-Directed Techniques

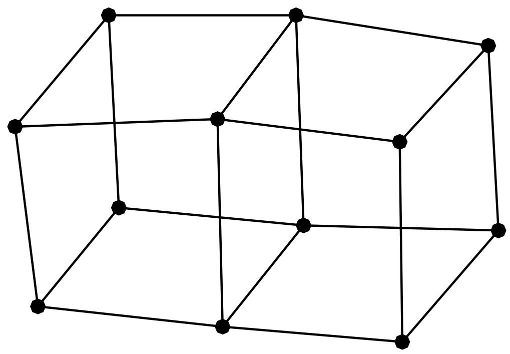
- [Kamada Kawai 89]
  - the forces try to place vertices so that their ***geometric distance*** in the drawing is equal to their ***graph-theoretic distance***
  - for each pair of vertices (u,v) use a ***spring*** with natural length dist(u,v)
- [Fruchterman Reingold 90]
  - system of forces similar to that of ***subatomic particles*** and celestial bodies
  - given drawing region acts as wall
  - ***n-body simulation***
- [Davidson Harel 89]
  - ***energy function*** takes into account vertex distribution, edge-lengths, and edge-crossings
  - given drawing region acts as wall
  - ***simulated annealing***

Examples

- drawings of the same graph constructed with the technique of [Davidson Harel 89] using three different energy functions







Advantages and Disadvantages of Force-Directed Techniques

Pro:

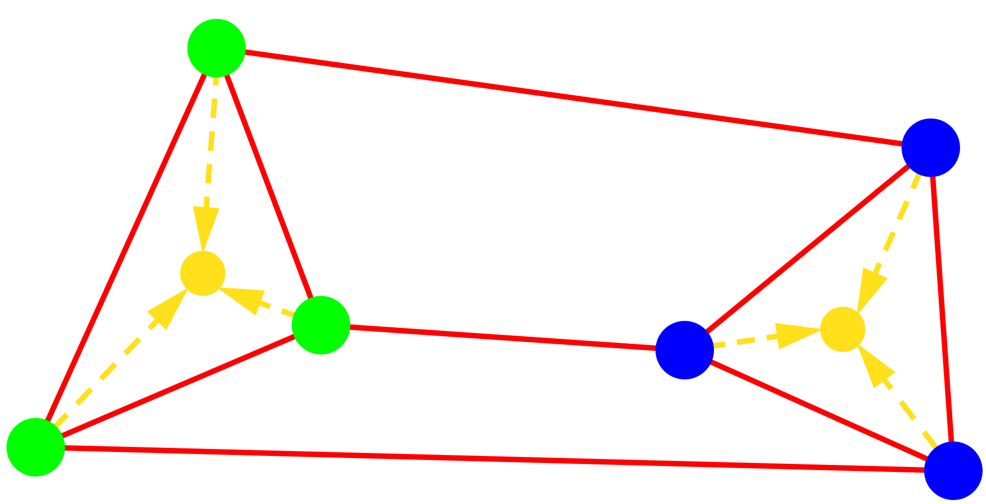
- relatively ***simple*** to implement
- ***heuristic improvements*** easily added
- ***smooth evolution*** of the drawing into the final configuration helps preserving the user's mental map
- ***can be extended to 3D***
- often able to detect and display ***symmetries***
- ***works well*** in practice ***for small graphs*** with regular structure
- limted ***constraint satisfaction*** capability

Con:

- ***slow*** running time
- ***few theoretical results*** on the quality of the drawings produced
- ***diffcult to extend*** to orthogonal and polyline drawings
- ***limited constraint satisfaction*** capability

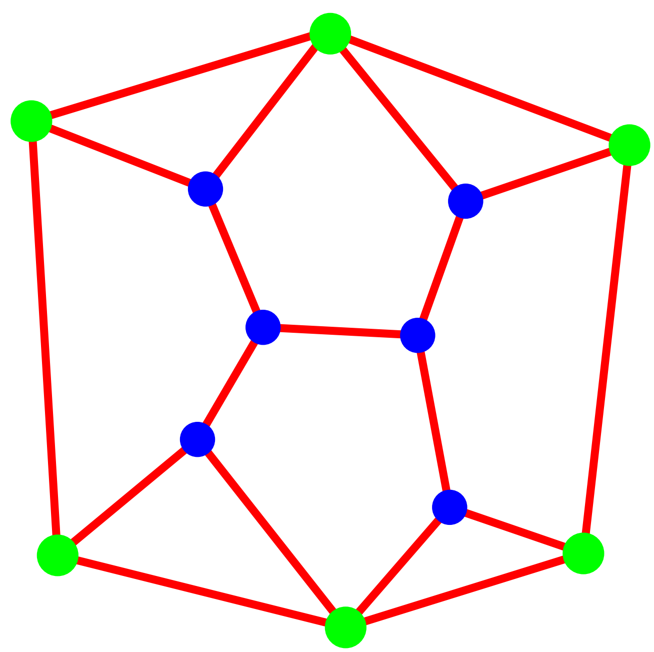
Constraints in Force-Directed Techniques

- ***position constraints*** can be easily imposed
  - we can constrain each vertex to remain in a prescribed region
- other ***constraints*** can be satisfied provided they can be ***expressed by means of forces***, e.g,
  - "***magnetic field***" to impose orientation constraints [Sugiyama Misue 84]
  - dummy "***attractor***" vertex to enforce grouping



Springs for Planar Graphs

- use springs with natural length 0, and attractive force proportional to the length
- pin down the vertices of the ***external face*** to form a given ***convex polygon*** (position constraints)
- let the system go ...



- the final configuration is a state of minimum energy: $\min \sum_{e} [\mathrm{length}(e)]^2$
- equivalent to the ***barycentric mapping*** [Tutte 60]:

$$\mathbf{p}(v) = 1/\deg(v) \sum_{(v,w)} \mathbf{p}(w)$$

General Directed Graphs

Layering Method for Drawing General Directed Graphs

- ***Layer assignment:*** assign vertices to layers trying to minimize
  - ***edge dilation***
  - ***feedback edges***
- ***Placement:*** arrange vertices on each layer trying to minimize
  - ***crossings***
- ***Routing:*** route edges trying to minimize
  - ***bends***
- ***Fine tuning:*** improve the drawing with local modifications

[Carpano 80]

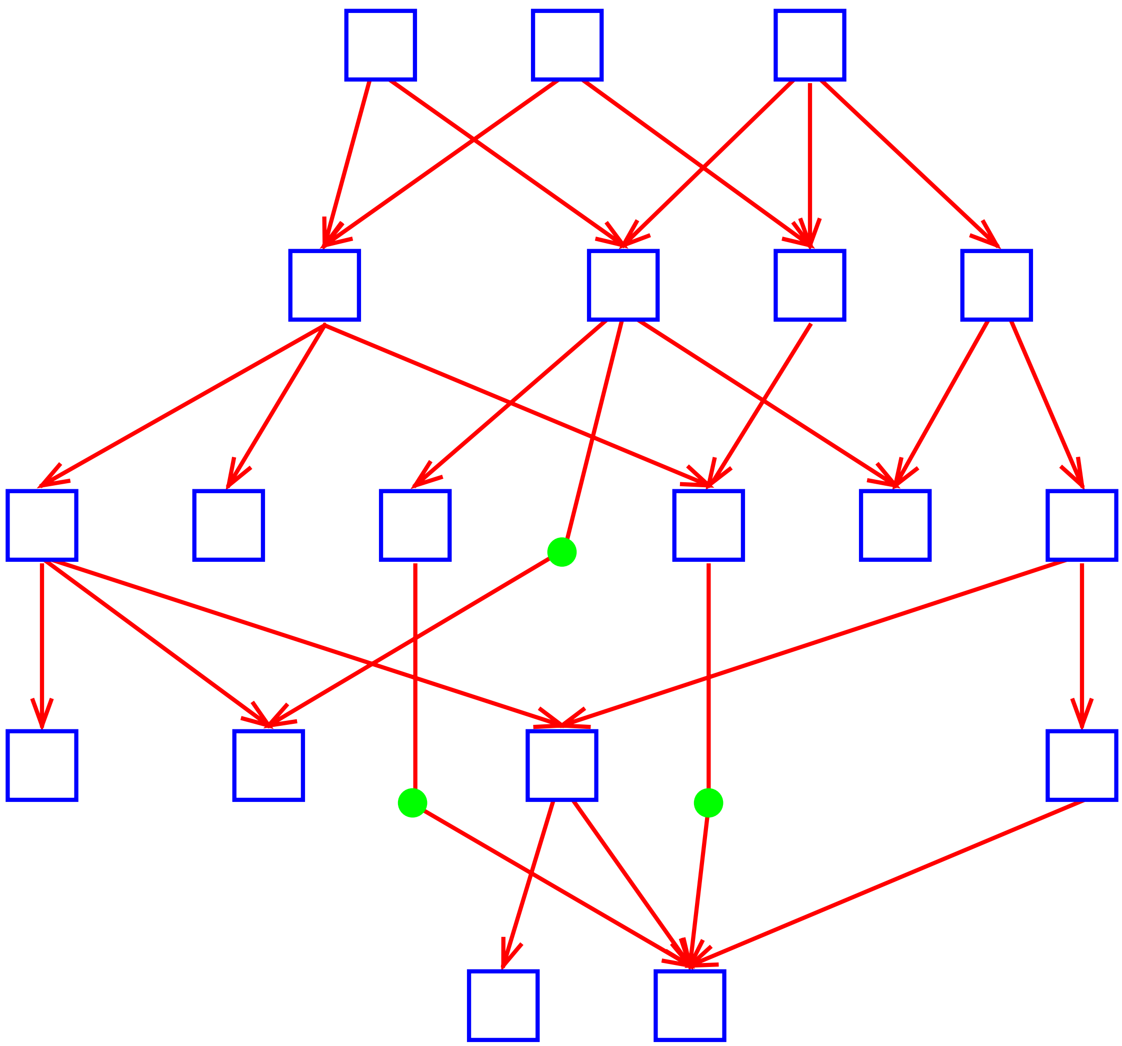
[Sugiyama Tagawa Toda 81]

[Rowe Messinger et al. 87]

[Gansner North 88]

Example

- [Sugiyama Tagawa Toda 81]



Declarative Approaches

Declarative Approach

- **These approaches cover a broad range of possibilities:**
  - **Tightly-coupled**: specification and algorithms cannot be separated from each other.
  - **Loosely coupled**: the specification language is a separate module from the algorithms module.
  - Most of the approaches are somewhere in between ...

Tightly-coupled approaches

Advantages:

- The algorithms can be optimized for the particular specification.
- The problem is well-defined.

Disadvantages:

- Takes an expert to modify the code (difficult extensibility).
- User has less flexibility.

Loosely-coupled approaches

Advantages:

- Flexible: the user specifies the drawing using constraints, and the graph drawing module executes it.
- Extensible: progressive changes can be made to the specification module and to the algorithms module.

Disadvantages:

- Potential “impedance mismatch” between the two modules.
- Efficiency: more difficult to guarantee.

Languages for Specifying Constraints

- Languages for display specification
  - ThingLab [Borning 81]
  - IDEAL [Van Wyk 82]
  - Trip [Kamada 89]
  - GVL [Graham & Cordy 90]
- Grammars
  - Visual Grammars [Lakin 87]
  - Picture Grammars [Golin and Reiss 90]
  - Attribute Grammars [Zinßmeister 93]
  - Layout Graph Grammars [Brandenburg94] [Hickl94]
  - Relational Grammars [Weitzman &Wittenburg 94]
- Visual Constraints
  - U-term language [Cruz 93]
  - Sketching [Gleicher 93] [Gross94]

Visual

Used in GD

Used in GD and Visual

ThingLab [Borning 81]

- Graphical objects are defined by example, and have a ***typical*** part and a ***default*** part.
- Constraints are associated with the classes (methods specify constraint satisfaction).
- Object-oriented (message passing, inheritance).
- Visual programming language.

Ideal [Van Wyk 82]

- Textual specification of constraints.
- Graphical objects are obtained by instantiating
  abstract data types, and adding constraints.
- Uses complex numbers to specify coordinates.

GVL [Graham & Cordy 90]

- Visual language to specify the display of program data structures.
- Pictures can be specified ***recursively*** (the display of a linked list is the display of the first element of the list, followed by the display of the rest of the list.

Layout Graph Grammars

[Brandenburg 94] [Hickl 94]

- grammatical (rule-based method) for drawing graphs
- extension of a ***context-free string grammar***
  - underlying context-free graph grammar
  - layout specification for its productions
- by repeated applications of its productions, a graph grammar generates labeled graphs, which define its graph language
- class of layout graph grammars for which optimal graph drawings can be constructed in polynomial time:
  - H-tree layouts of complete binary trees
  - hv-drawings of binary trees
  - series-parallel graphs
  - NFA state transition diagrams from regular expressions

Picture Grammars

[Golin & Reiss 90, Golin 91]

- **Production rules use constraints.**
- **Terminals are:**
  - *shapes* (e.g., rectangle, circle, text)
  - *lines* (e.g., arrow)
- **spatial relationships between objects are *operators* in the grammar (e.g., over, left_of)**

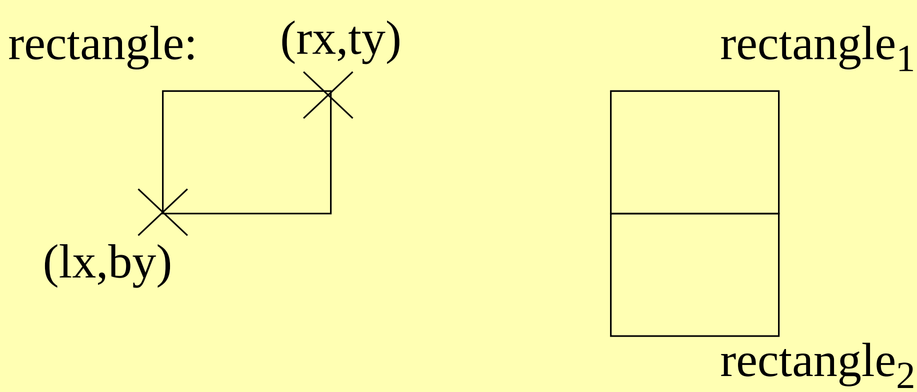
FIGURE $\rightarrow$ over (rectangle$_1$, rectangle$_2$)

Where

rectangle$_1$.lx == rectangle$_2$.lx

rectangle$_1$.rx == rectangle$_2$.rx

rectangle$_1$.by == rectangle$_2$.ty



- **More expressive relationships : *tiling*.**
- **Complexity of parsing has been studied.**

Relational Grammars

[Weitzman & Wittenburg 93, 94]

- **Generalization of attribute string grammars that allow for the specification of geometric positions in 2D and 3D, topological connectivity, arbitrary semantic relations holding among information objects.**

$Article \rightarrow Text\ Text\ Text\ Number\ Image$

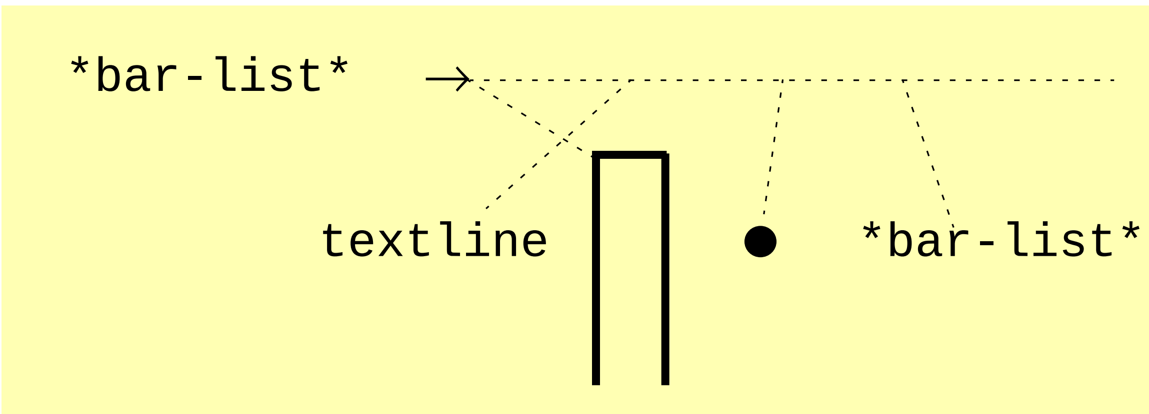
```
(Defrule (Make-Article The-Grammar)
         (0 Article)
         (1 Text)
          (2 Text (Author-Of   2 1))
          . . .

         :OUT
         (
            . . .

         (spaced-below 2 1)
         (spaced-below 3 1)
         (set-font 1 10pt :bold)
         (set-font 1  8pt :italic)

          . . .
                                   ))
```

- **Constraints are solved with DeltaBlue (U. of Washington) for non-cyclic constraints.**

Visual Grammars

[Lakin 87]

- **Contex-free grammar.**
- **Symbols are visual, and are visually annotated.**



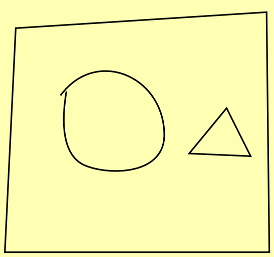
- **The interpretation of the visual symbols is left to the implementation.**

Expressing Constraints by Sketching

- **Briar [Gleicher 93]**

  Constraint-based drawing program:
  - Direct manipulation drawing techniques.
  - Makes relationships between graphical objects persistent
  - Performance concerns in solving constraints.

- **Spatial Relation Predicates [Gross 94]**



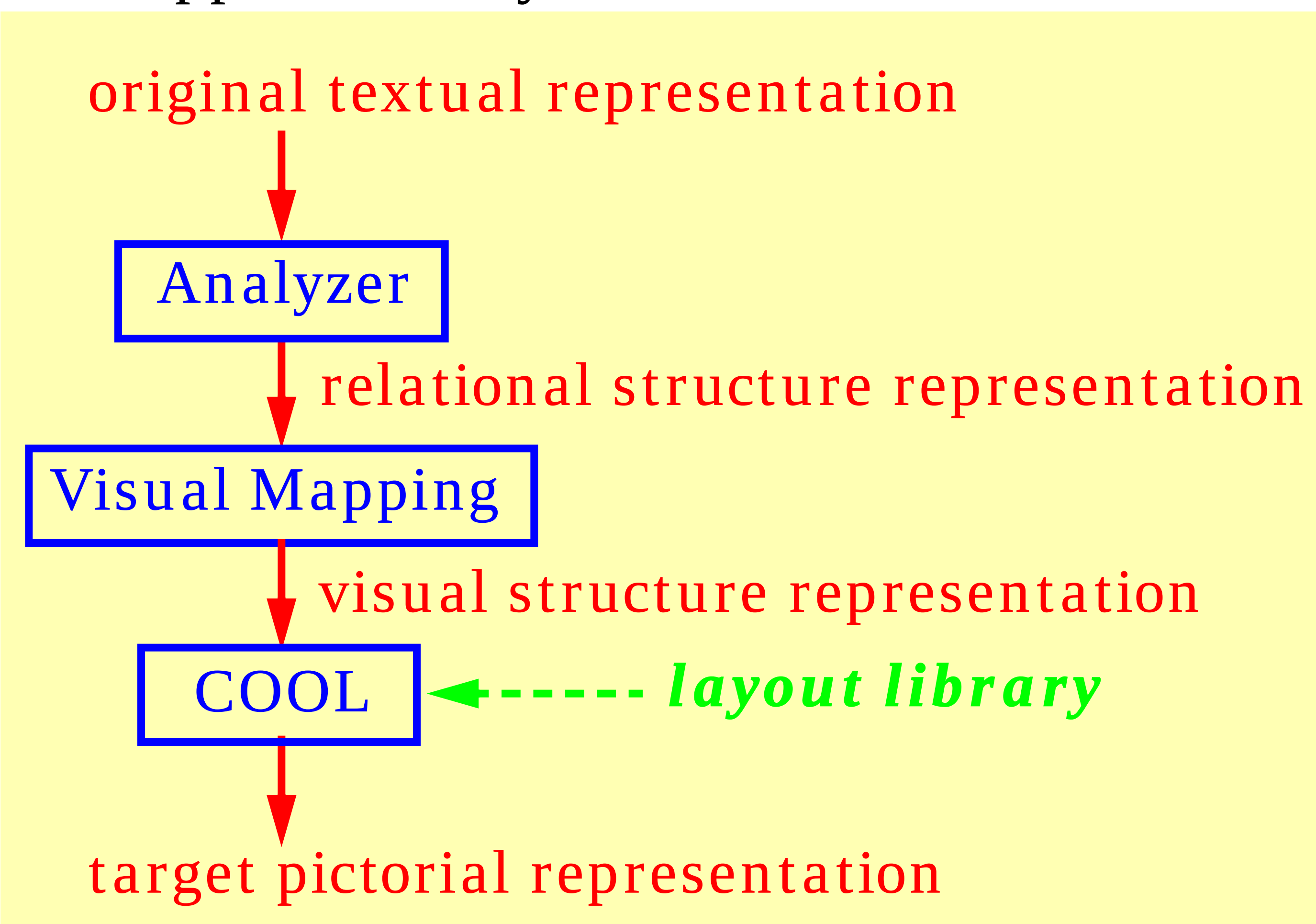
(CONTAINS BOX CIRCLE)
(CONTAINS BOX TRIANGLE)
(IMMEDIATELY-RIGHT-OF CIRCLE TRIANGLE)
(SAME-SIZE CIRCLE TRIANGLE)

  - Applications include retrieval of buildings from an architecture database.

COOL

[Kamada 89]

- framework for visualizing abstract objects and relations.
- constraint-based object layout system
  - ***rigid*** constraints
  - ***pliable*** constraints
  - conflicting constraints can be solved approximately



ANDD

[Marks et al]

- layout-aesthetic concerns subordinated to perceptual-organizational concerns
- notation for describing the visual organization of a network diagram
  - alignment, zoning, symmetry, T-shape, hub shape
- layout task as a constrained optimization problem:
  - constraints derived from a visual-organization specification
  - optimality criteria derived from layout-aesthetic considerations
- two heuristic algorithms:
  - rule-based strategy
  - massive parallel genetic algorithm

Visual Graph Drawing

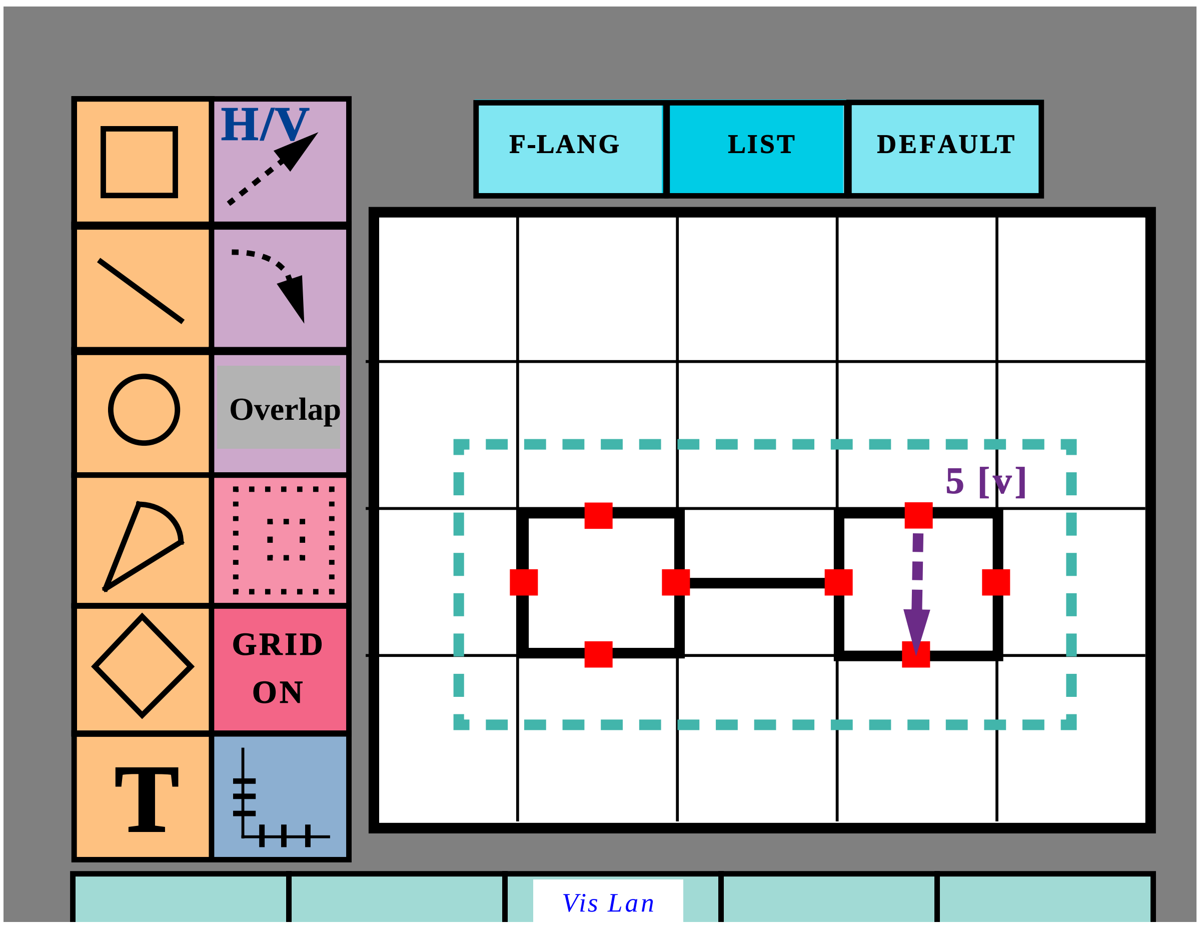
[Cruz, Tamassia Van Hentenryck 93]

- a ***visual*** approach to graph drawing can reconcile ***expressiveness*** with ***efficiency***
- **Goals**
  - ***Visual*** specification of layout ***constraints***: the user should not have to type a long list of textual specifications
  - ***Visual*** specification of aesthetic criteria associated with ***optimization*** problems
  - ***Extensibility***: the user should not be limited to a prespecified set of visual representations.
  - ***Flexibility***: the user should not have to give precise geometric specifications.

U-term Language

[Cruz 93, 94]

- **Visual constraints.**
- **Simplicity and genericity of the basic constructs.**
- **Ability to specify a variety of displays: graphs, higraphs, bar charts, pie charts, plot charts, ...**
- **Compatibility with the framework of an object-oriented database language, DOODLE.**
- **Recursive visual specification.**

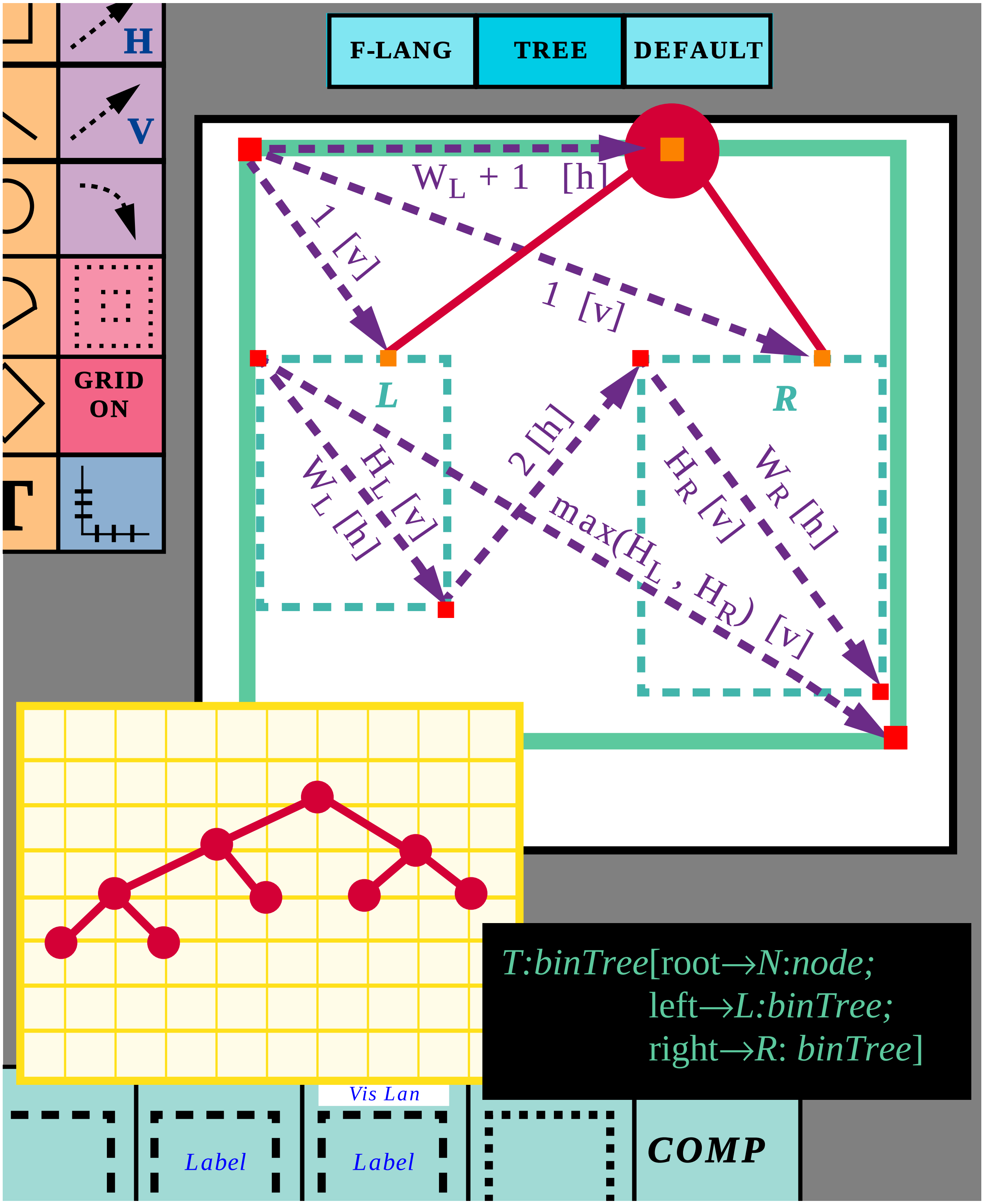


Efficient Visual Graph Drawing

[Cruz Garg 94] [Cruz Garg Tamassia 95]

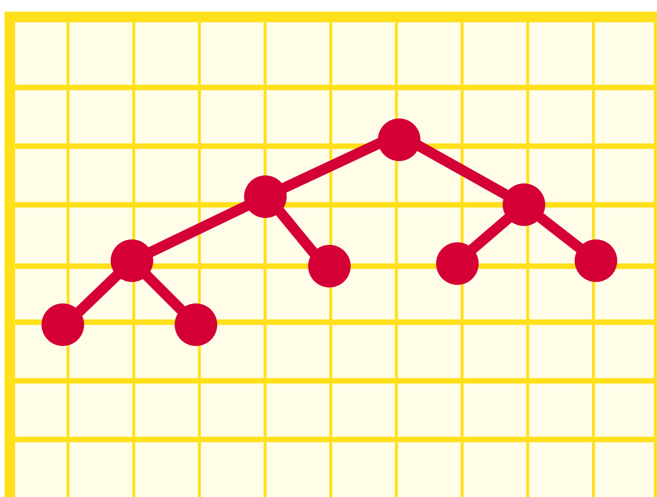
- graph stored in an object-oriented database
- drawing defined "***by picture***" using recursive visual rules of the language DOODLE [Cruz 92]
- a set of ***constraints*** is generated by the application of the visual rules to the input graph
- various types of drawings can be visually expressed in such a way that the resulting set of constraints can be solved in ***linear time***, e.g.,
    - drawings of trees (upward drawings, box inclusion drawings)
    - drawings of series-parallel digraphs (delta drawings)
    - drawings of planar acyclic digraphs (visibility drawings, upward planar polyline drawings)

Tree Layout

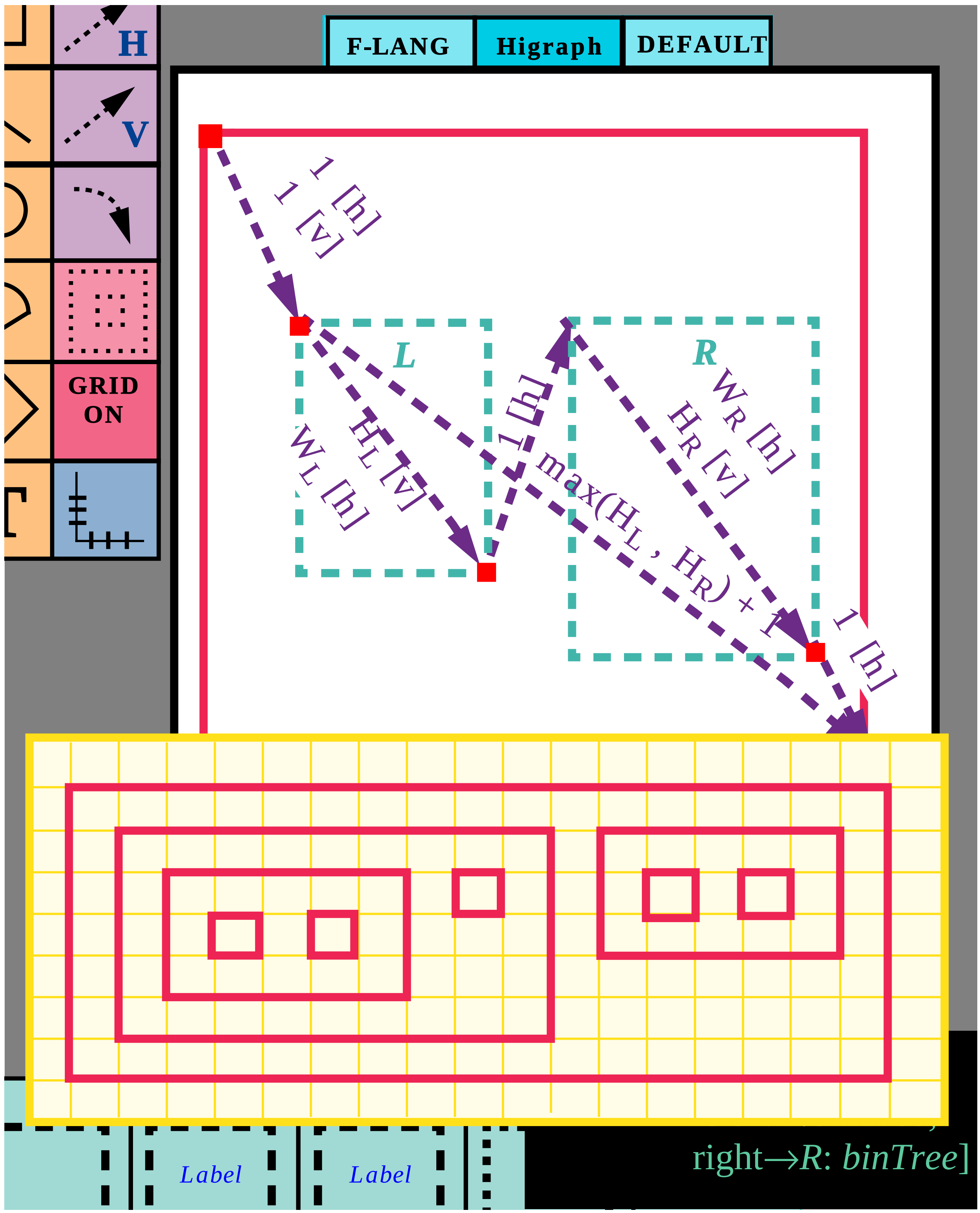

F-LANG
TREE
DEFAULT
H
V
GRID ON
W_L + 1 [h]
1 [v]
1 [v]
L
R
2 [h]
W_L [h]
H_L [v]
H_R [v]
W_R [h]
max(H_L , H_R) [v]
T:binTree[root→N:node;
left→L:binTree;
right→R: binTree]
Vis Lan
Label
Label
COMP

Characteristics of the Previous Tree Drawings

- Level Drawings
  - Upward
  - Planar
  - Nodes at the same distance from the root are horizontally aligned.
- Display of symmetries.
- Display of isomorphic subtrees.



Change a few things . . .



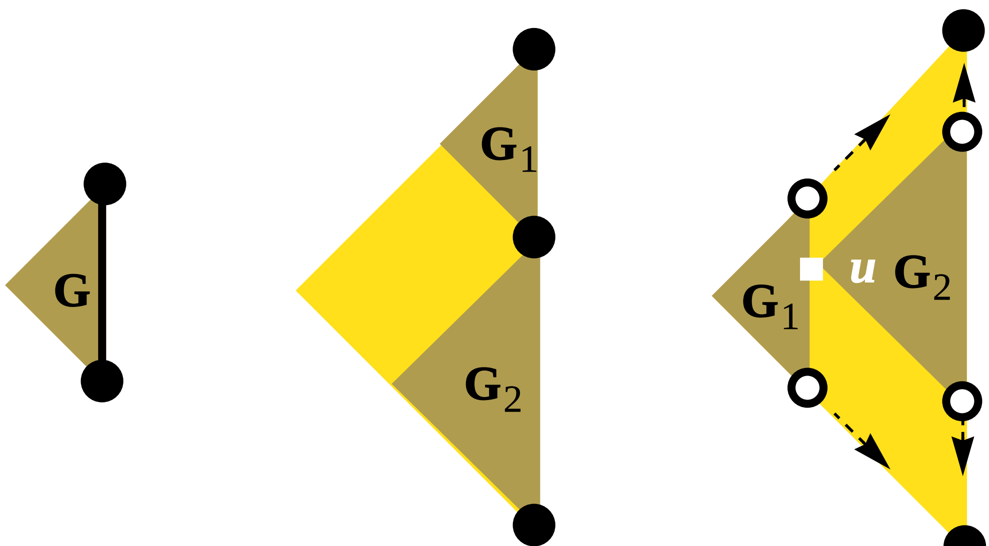
Efficient Visual Graph Drawing

[Cruz & Garg 94]

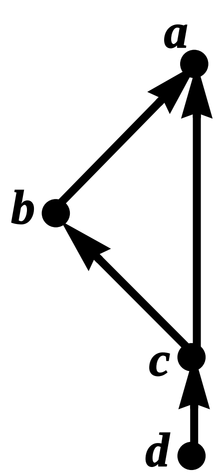
- **Recognize classes of graphs and drawings that can be expressed with DOODLE and evaluated efficiently.**
- **Devise algorithms and data structures for performing drawings in linear time (optimal time):**
  - Trees (upward drawing, box inclusion drawing).
  - Series-parallel digraphs (delta drawing).
  - Planar acyclic digraphs (visibility drawing, upward planar polyline drawing).
- **Next:**
  - Extend above results to other classes of graphs and drawings.
  - Constraint viewpoint: framework for evaluating constraints efficiently.
  - Incorporate these algorithms into a declarative graph drawing system that uses **DOODLE**.

More examples

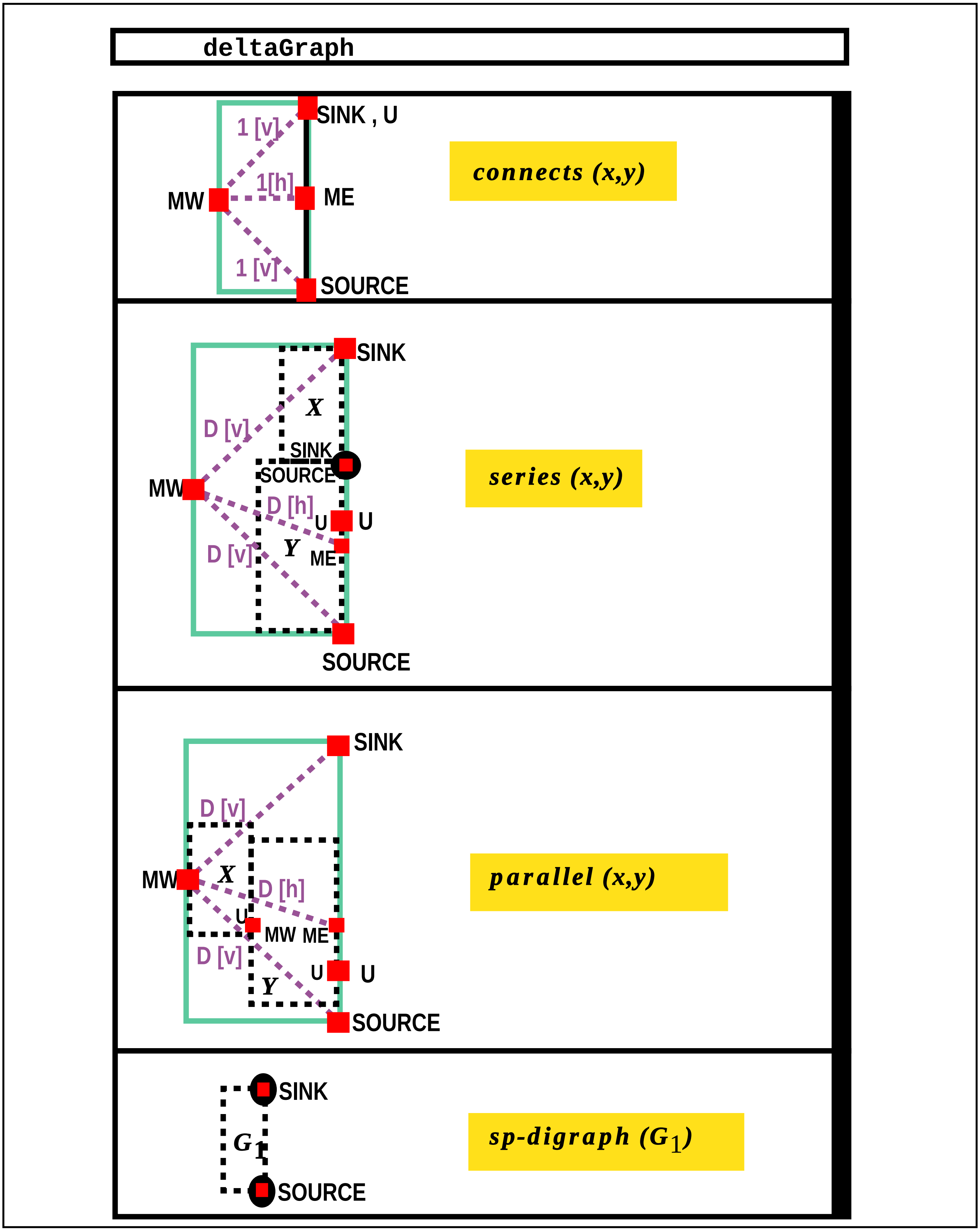
- Series-parallel graphs / delta-drawings [Bertolazzi, Cohen, Di Battista, Tamassia & Tollis, 92]



Base case ***Series composition*** ***Parallel composition***

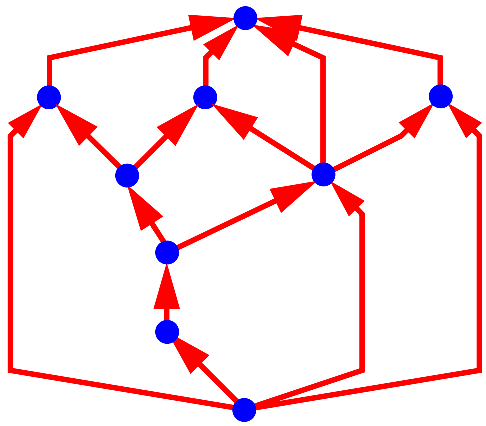


Example

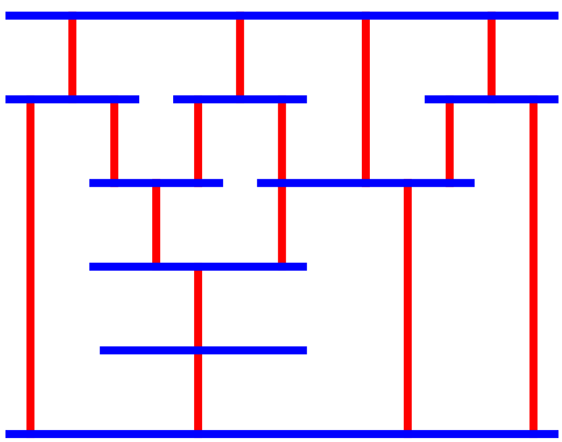

deltaGraph
SINK , U
1 [v]
1[h]
MW
ME
1 [v]
SOURCE
connects (x,y)
SINK
X
D [v]
SINK
SOURCE
MW
D [h]
U
U
D [v]
Y
ME
SOURCE
series (x,y)
SINK
D [v]
MW
X
D [h]
U
MW
ME
D [v]
U
U
Y
SOURCE
parallel (x,y)
SINK
G1
SOURCE
sp-digraph (G1)

Drawings of Planar DAGs

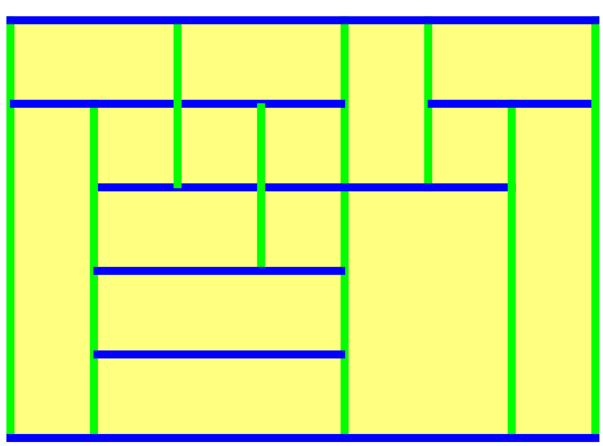
- planar upward drawing



- visibility drawing



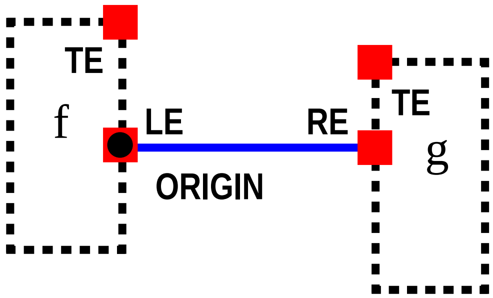
- tessellation drawing



Tessellation Drawing

TessellationDrawing

v: sourceVertex

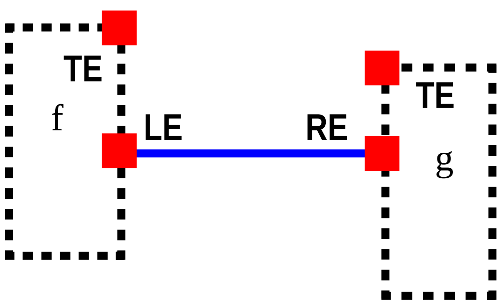


F-Language

v: sourceVertex [leftFace → f : face ;
rightFace → g: face]

TessellationDrawing

v: vertex

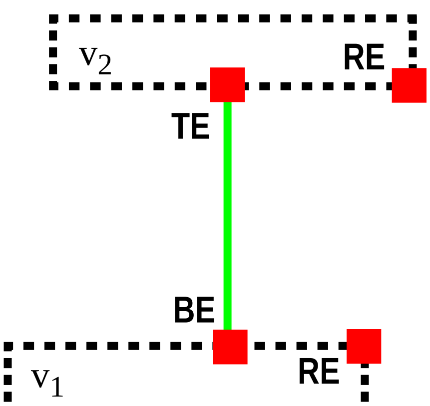


F-Language

v: vertex [leftFace → f : face ;
rightFace → g: face]

TessellationDrawing

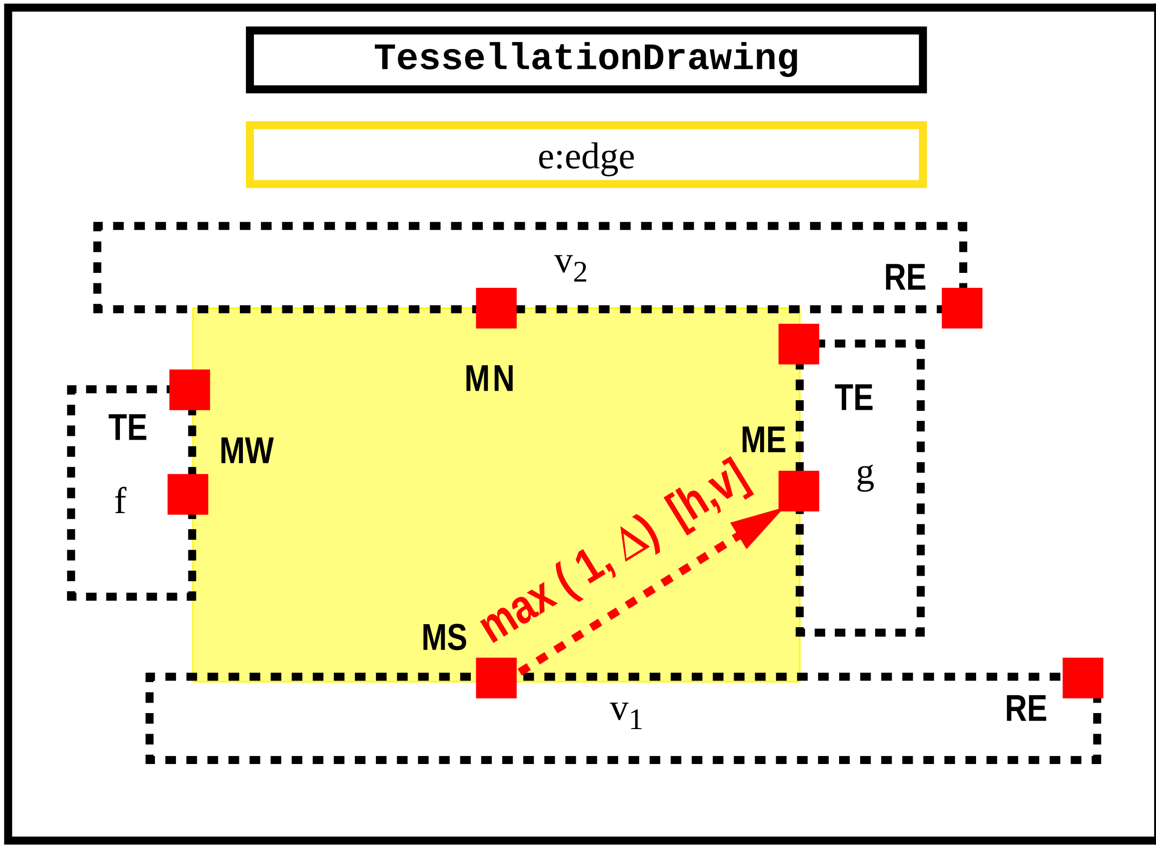
f: face



F-Language

f: face [α→ v_2: vertex ;
bottomVertex → v_1: vertex]

Tessellation Drawing



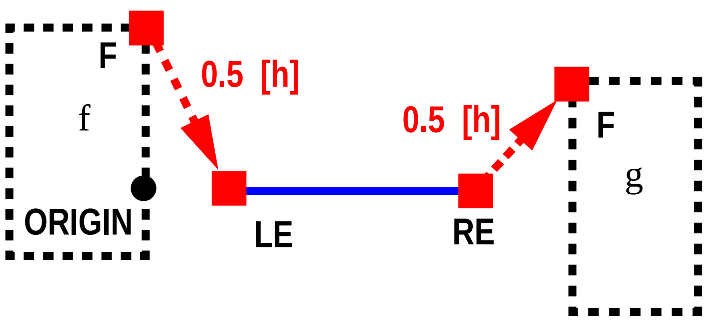
F-Language

e: edge [from $\rightarrow v_1$: vertex;
to $\rightarrow v_2$: vertex;
leftFace $\rightarrow$ f: face;
rightFace $\rightarrow$g: face]

Visibility Drawing

VisibilityDrawing

v: sourceVertex

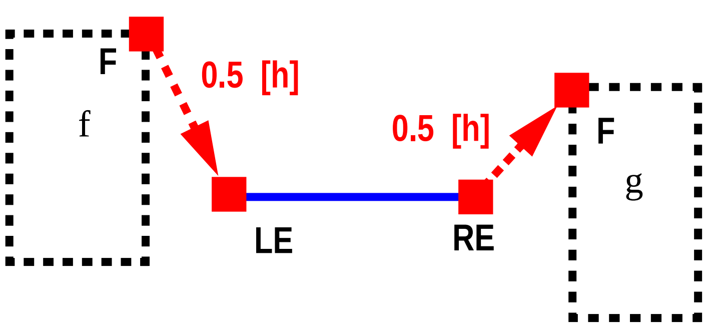


F-Language

v: sourceVertex [leftFace $\rightarrow$ f : face ;
rightFace $\rightarrow$ g: face]

VisibilityDrawing

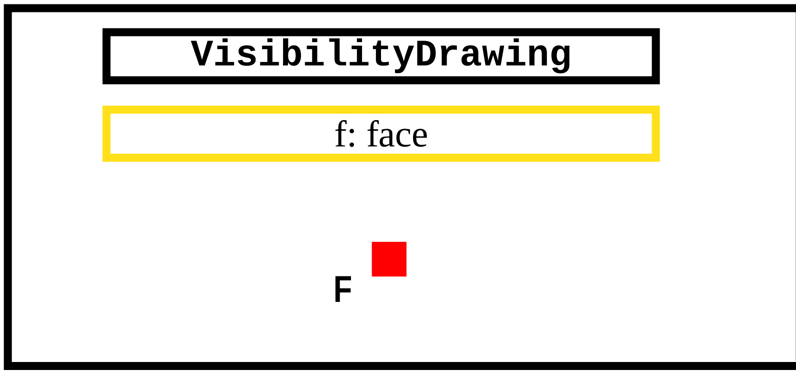
v: vertex



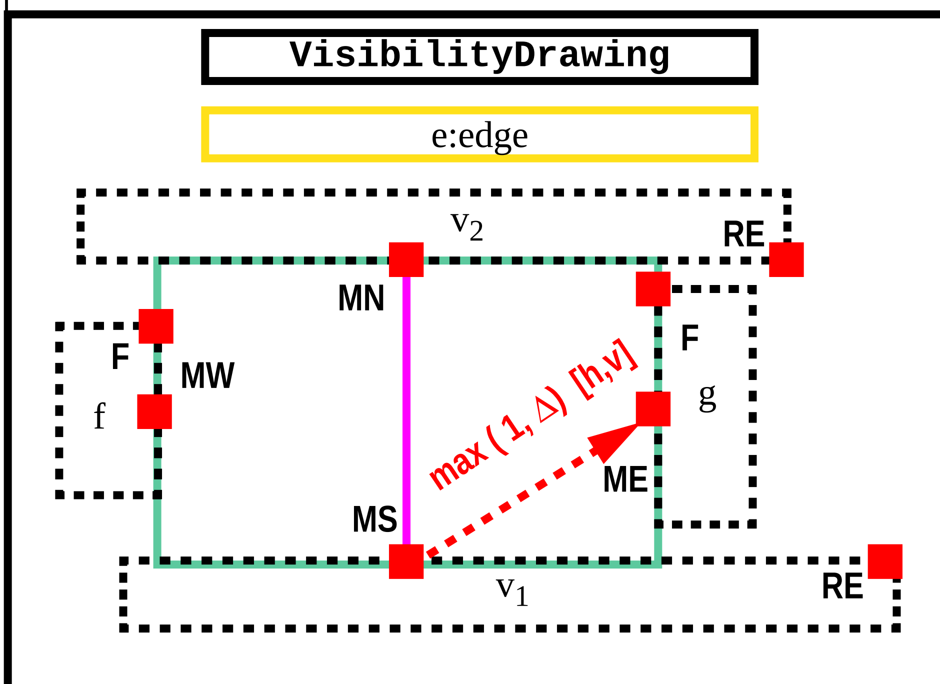
F-Language

v: vertex [leftFace $\rightarrow$ f : face ;
rightFace $\rightarrow$ g: face]

Visibility Drawing



f: face



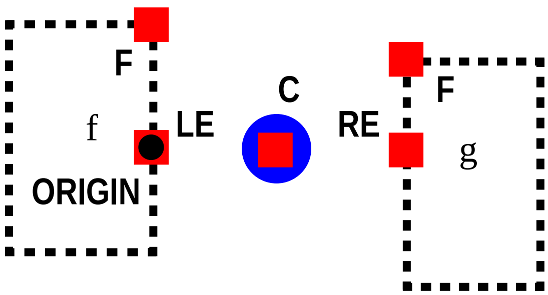
F-Language

e: edge [from $\rightarrow v_1$: vertex;
to $\rightarrow v_2$: vertex;
leftFace $\rightarrow$ f: face;
rightFace $\rightarrow$g: face]

Upward Polyline Drawing

PolylineDrawing

v: sourceVertex

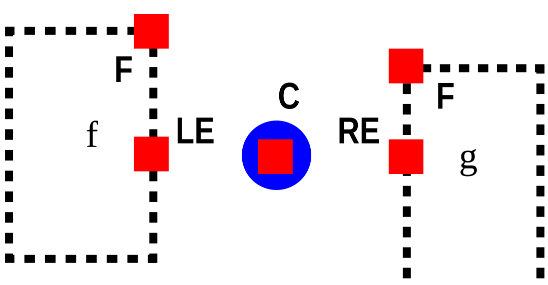


F-Language

v: sourceVertex [leftFace ⟶ f : face ;
rightFace ⟶ g: face]

PolylineDrawing

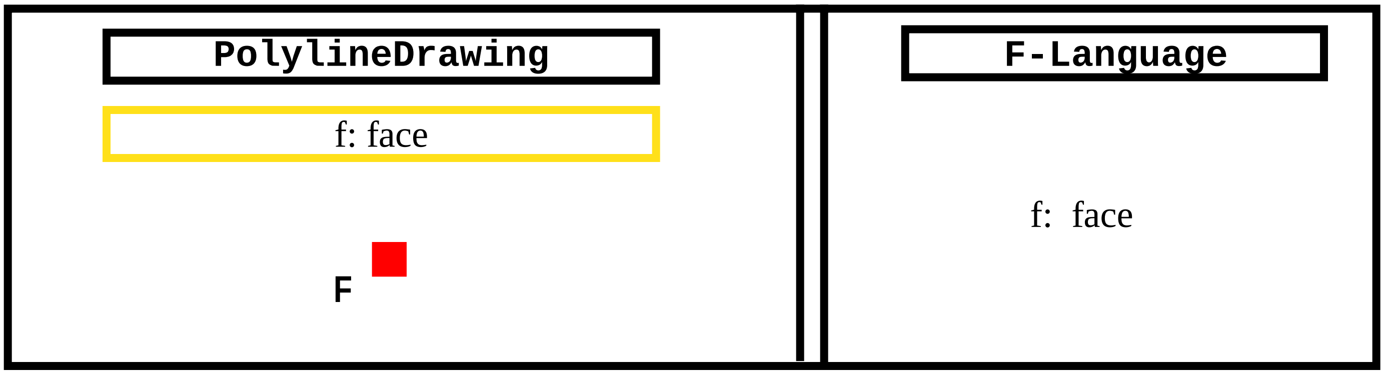
v: vertex

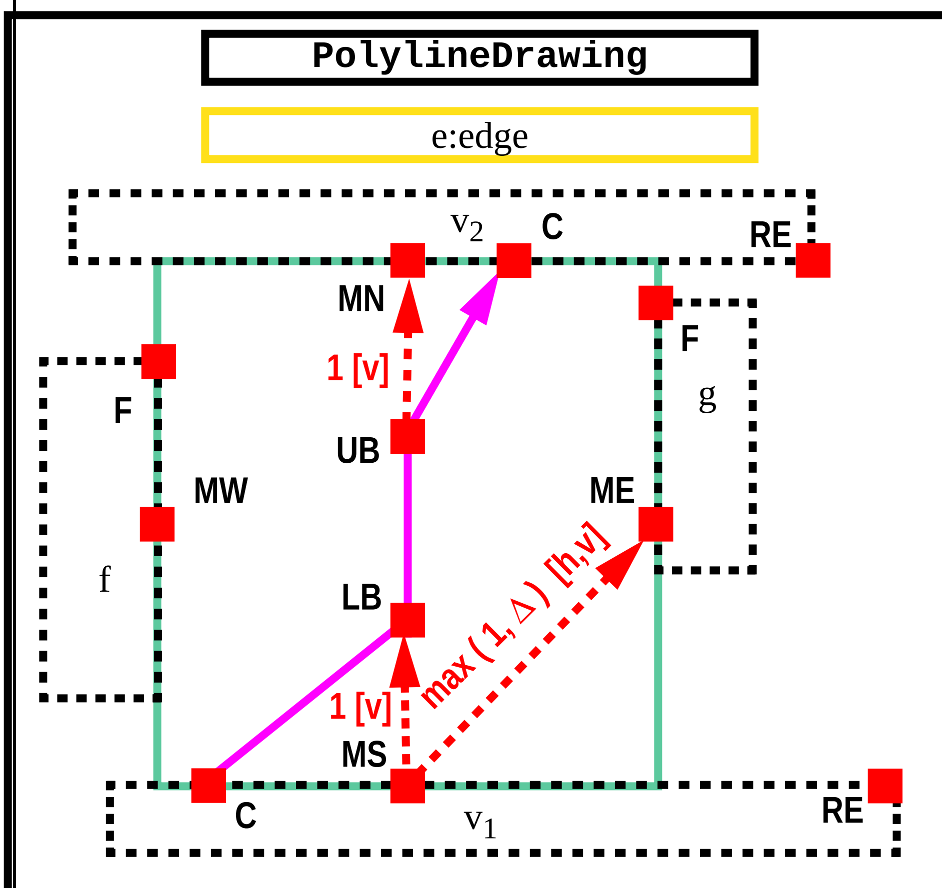


F-Language

v: vertex [leftFace ⟶ f : face ;
rightFace ⟶ g: face]

Upward Polyline Drawing





F-Language

e: edge [from $\rightarrow v_1$: vertex;
to $\rightarrow v_2$: vertex;
leftFace $\rightarrow$ f: face;
rightFace $\rightarrow$ g: face]

Challenges and Open Problems (Declarative Approach):

- New approach, therefore much left to explore, in particular:
  - New specification languages.
  - Reducing the "impedance mismatch."
  - Design of user interfaces, and evaluation in different environments/ applications.
  - Identification of levels of complexity in drawing graphs (e.g., with graph grammars, constraint languages).
  - Expressiveness of the specification languages, in particular of declarative and visual languages.
  - Refinement of the *diagram server* hierarchy, so that we can have a true "tool box" for the declarative, loosely-coupled approach.

Systems

Some Graph Drawing Systems

- ***Graph Drawing Server***
  (Brown University, USA)
  - loki.cs.brown.edu:8081/graphserver/
  - Roberto Tamassia (rt@cs.brown.edu)
- ***GDToolkit***
  (University of Rome III)
  - www.dia.uniroma3.it/people/gdb/wp12/GDT.html
  - Giuseppe Di Battista (dibattista@iasi.rm.cnr.it)
- ***Graphlet***
  (University of Passau, Germany)
  - www.fmi.uni-passau.de/Graphlet/
  - Michael Himsolt (himsolt@fmi.uni-passau.de)
- ***GraphViz***
  (AT&T Research)
  - www.research.att.com/sw/tools/graphviz/
  - Sthephen North (north@research.att.com)