Graph Drawing
Algorithmic and Declarative Approaches

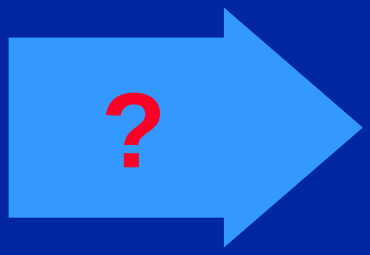
Peter Eades
University of Newcastle, Australia
University of Limerick, Ireland

Graph Drawing

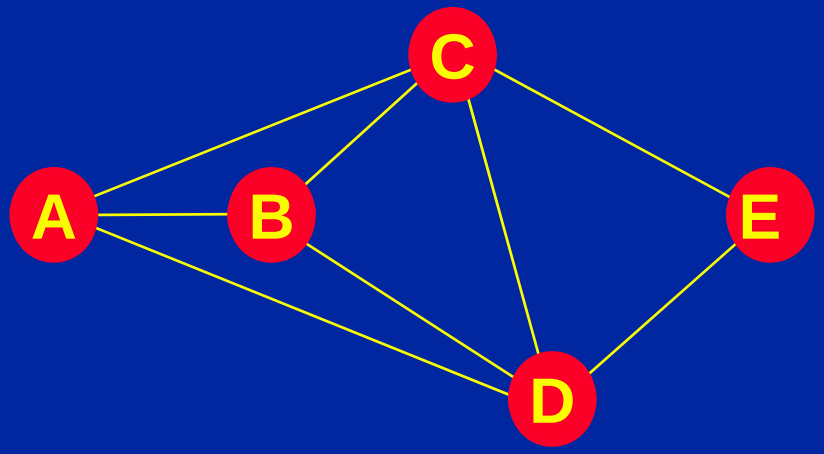
The classical graph drawing problem is to develop algorithms to draw graphs.

*The input is a graph with **no geometry***

A - B, C, D
B - A, C, D
C - A, B, D, E
D - A, B, D, E
E - C, D



*The output is a drawing of the graph; the drawing should be **nice***



Tokyo subway map

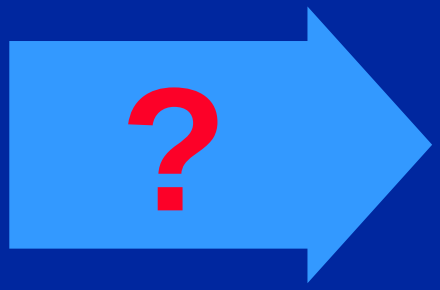
psfile

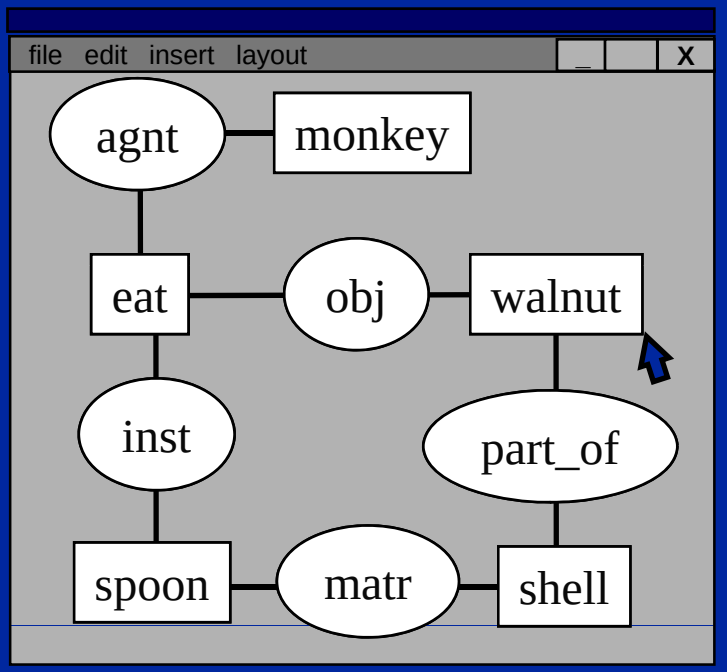
Graph Drawing

The output picture is used in a system design/analysis tool.

The input graph is some relational information.

```
agnt(monkey, eat).
inst(eat, spoon).
obj(eat, walnut).
part_of(walnut, shell).
matr(spoon, shell).
```





The graph drawing problem is to design techniques to give good drawings of graphs.

Approaches

A graph drawing function is an optimization algorithm for specific optimization goals.

A graph drawing function consists of a set of rules for a drawing, and a generic method for obtaining a drawing that satisfies the rules.

Algorithmic Person

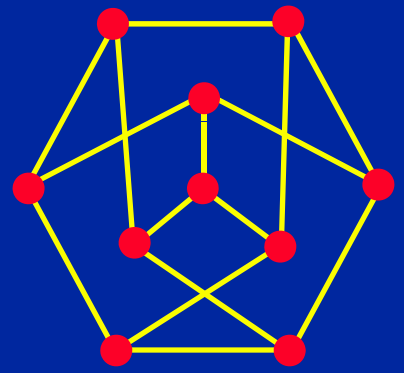
Declarative Person

Graph Drawing

1. User requirements

2. Established techniques

3. Algorithmic and Declarative Approaches



User requirements

There are 4 main requirements, from the user's point of view (Tao Lin 1992):

1. readability: the drawing should be easy to understand.

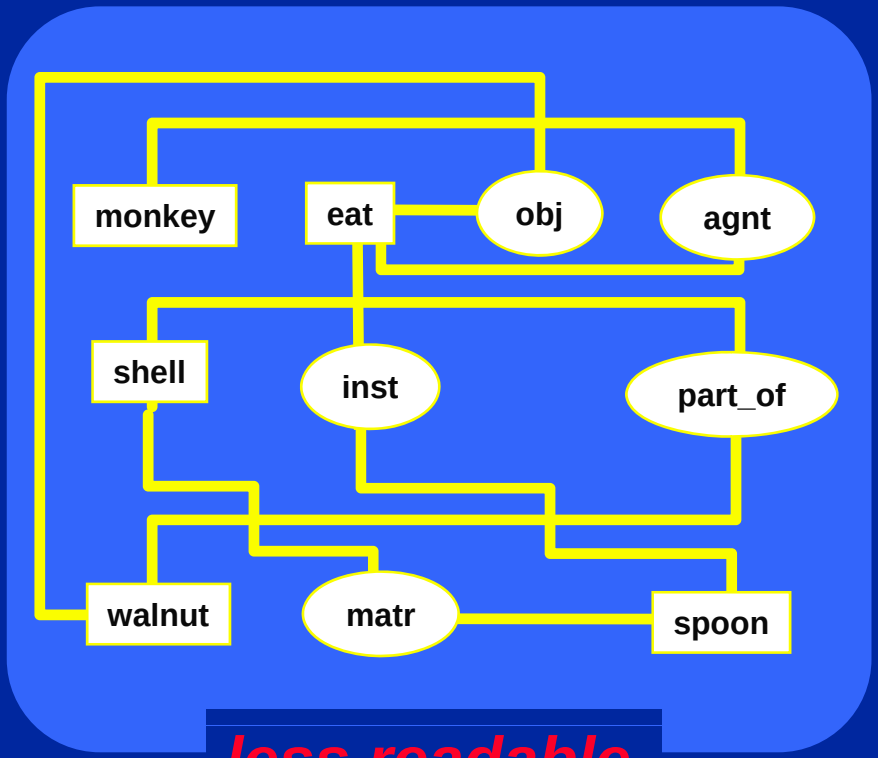
2. conformance: the drawing should conform to the diagrammatic conventions of the application.

3. controllability: specific users need to control the drawing functions.

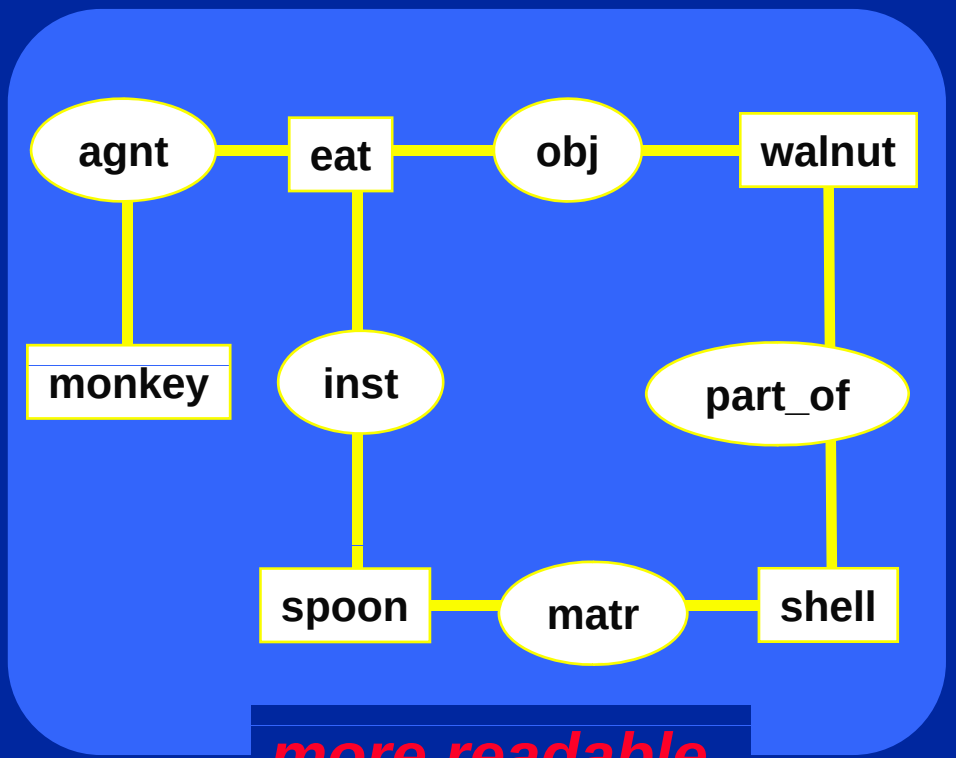
4. efficiency: the drawing functions should be efficient.

Readability

The drawing should be easy to read, easy to understand, easy to remember, beautiful.



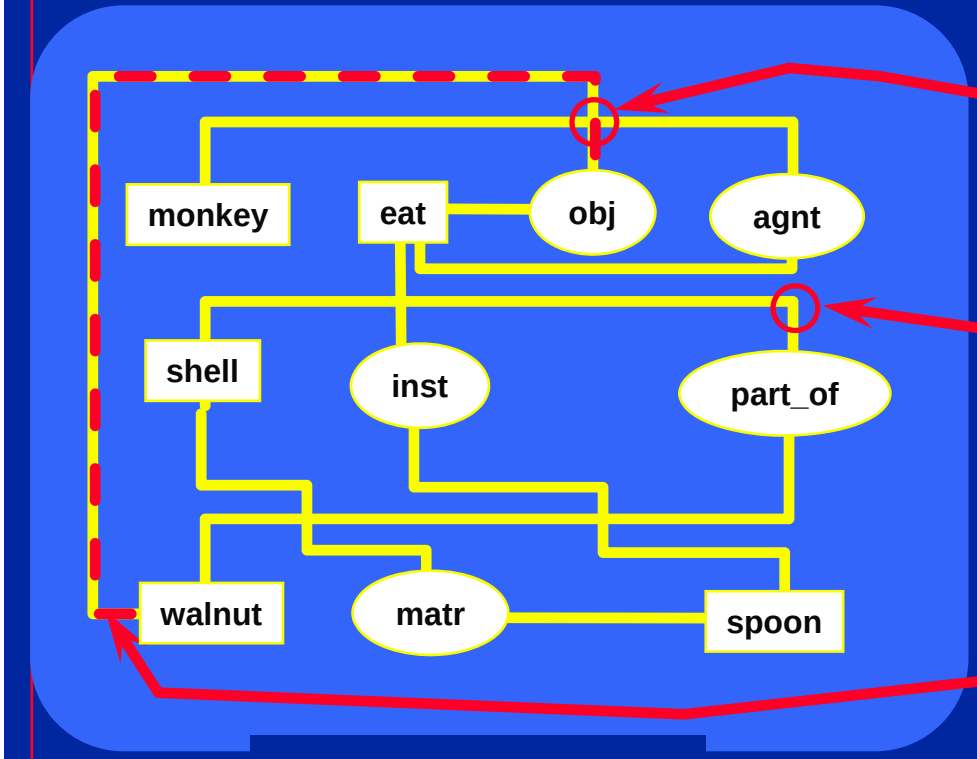
less readable



more readable

Readability

Some measures of readability are available:



Avoid edge ***crossings***

Avoid edge ***bends***

Avoid ***long*** edges

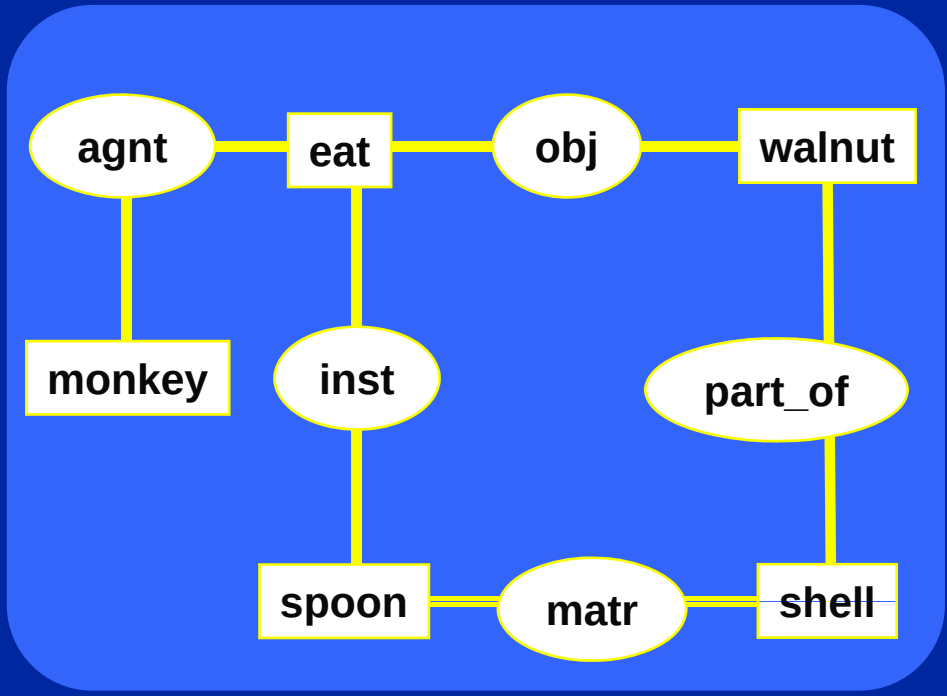
less readable

Readability

There are many readability criteria for good diagrams:

- minimum **edge crossings**,
- make edges **as straight as possible**,
- minimum **edge lengths**,
- maximum **resolution**,

and **many more**.



Conformance

The graph drawing must *conform* to the rules of the application domain. Specific domains have their own diagramming conventions. Sometimes, these have implications for graph drawing.

These are best expressed as constraints:

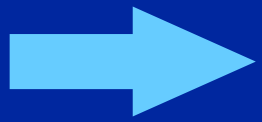
- *left_of(A,B)*: node A should appear to the left of node B.
- *above(A,B)*: node A should appear above node B.
- *close_to(A,B)*: node A should appear close to node B.
- *circle(A,B, …,)*: nodes A,B, … , should appear around in a circle.
- *horizontal(A,B)*: nodes A and B should share a horizontal line.

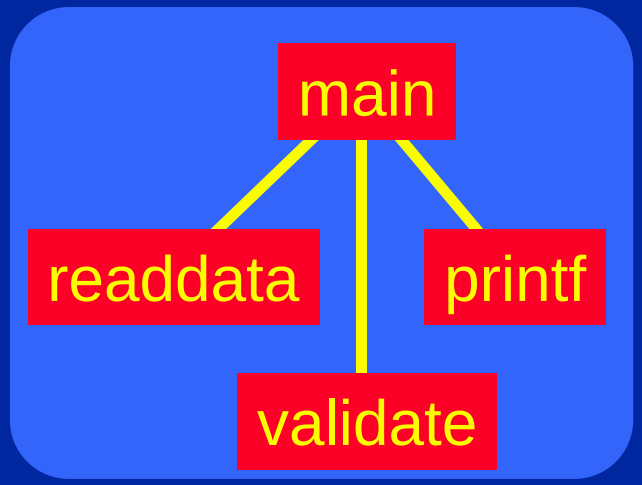
Conformance

Examples:

- In a call graph, the left-right order of nodes called from function A should be the same as the order of the calls in the source code.
- In a family tree females should be to the left of males.

```
main()
{
    readdata( … );
    validate( … );
    printf( … );
}
```





Controllability

Specific users at specific times need to control the layout of the graph.

Again, the user's control can be expressed as the imposition of constraints:

- *center(A)*: node A should appear near the center of the page.
- *closer(A,B,C)*: node A should be closer to C than B.

Constraints for user control are not very different from constraints for conformance. The main difference is that user control constraints cannot be hard coded.

Efficiency

Efficiency requirements vary:

- Interactive systems require a response within a fraction of a second.
- Users may wait for several minutes for large printed diagrams.

In terms of computational complexity, algorithms should have complexity $O(n^2)$.

Established techniques

- The Sugiyama algorithm
- TRIP
- Springs and cost function techniques

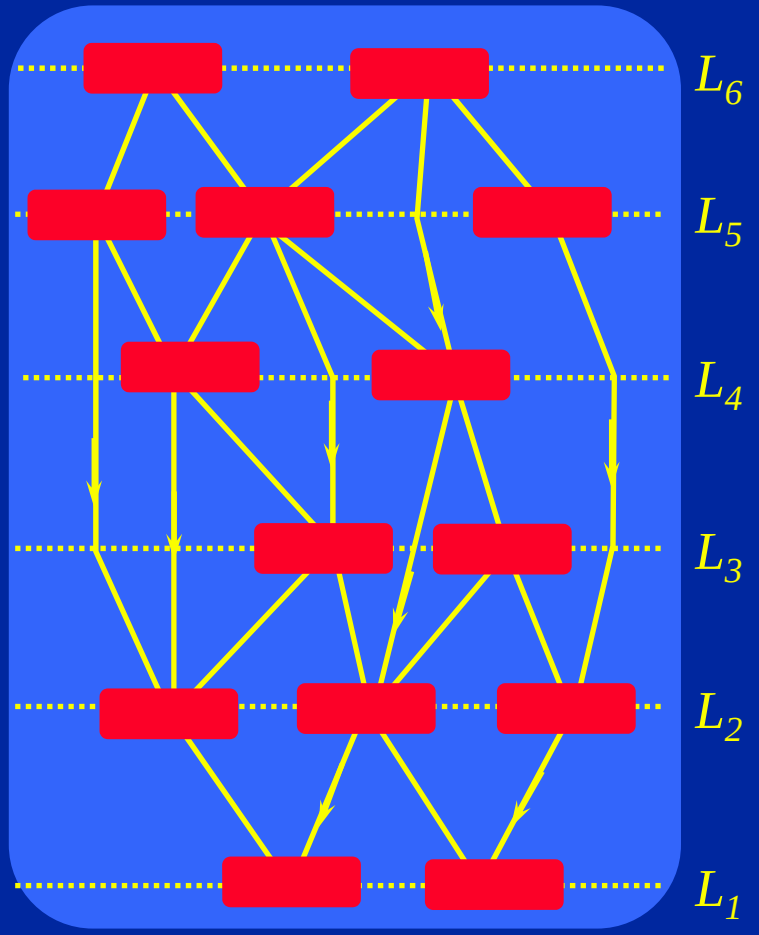
The Sugiyama algorithm

A *layered network* is a directed acyclic graph whose nodes are partitioned into *layers* L_1, L_2, …, L_k.

These networks are drawn so that:

- the nodes of layer L_m lie on the line $y=m$.
- edges are monotonic in the y direction.

The *flow* is represented from the *top to the bottom*.



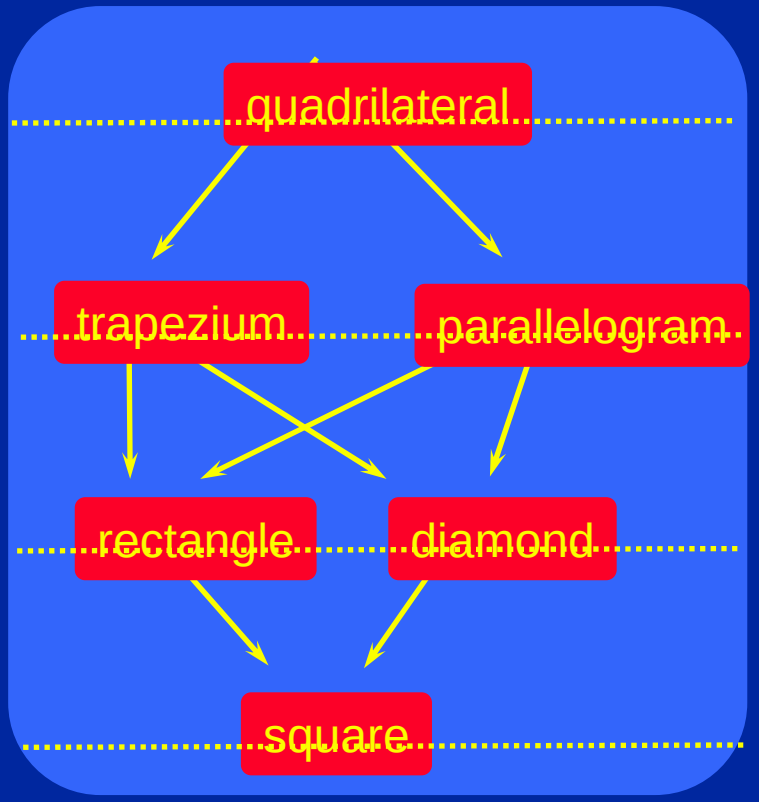
The Sugiyama algorithm

Layered networks are often used to represent dependency relations.

Sugiyama *et al.* developed a simple algorithm for drawing layered networks in 1979.

The algorithm aims for readability:

- few edge crossings
- edges as straight as possible
- nodes spread evenly over the page

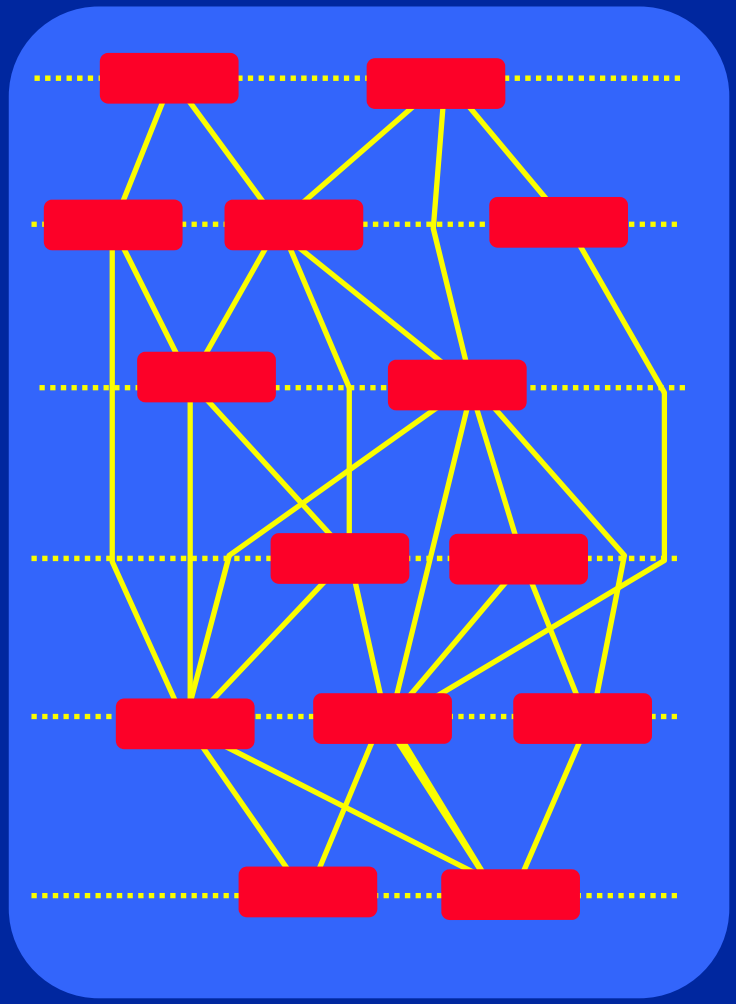


Sugiyama algorithm: 4 steps

1. *Eliminate directed cycles.*
2. *Place each node into a layer.*
3. *Order the nodes within each layer to avoid edge crossings.*
4. *Straighten the long edges.*

Each step has a corresponding optimization problem.

We will just look at steps 2 and 3.



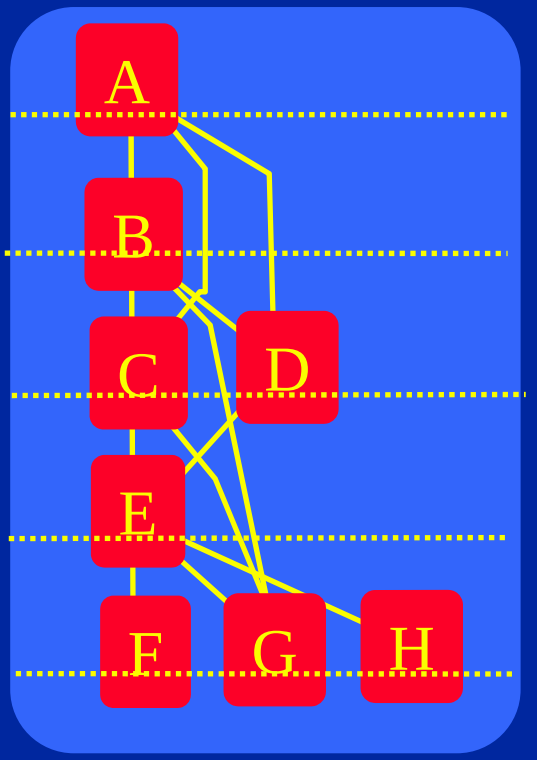
Sugiyama Step 2: Layering

Step 2

- Input: an acyclic directed graph
- Output: an assignment of nodes to layers, such that all edges are directed downward.

A - B, C, D
B - C, D, G
C - E, G
D - E
E - F, G, H
F -
G -
H -

step 2



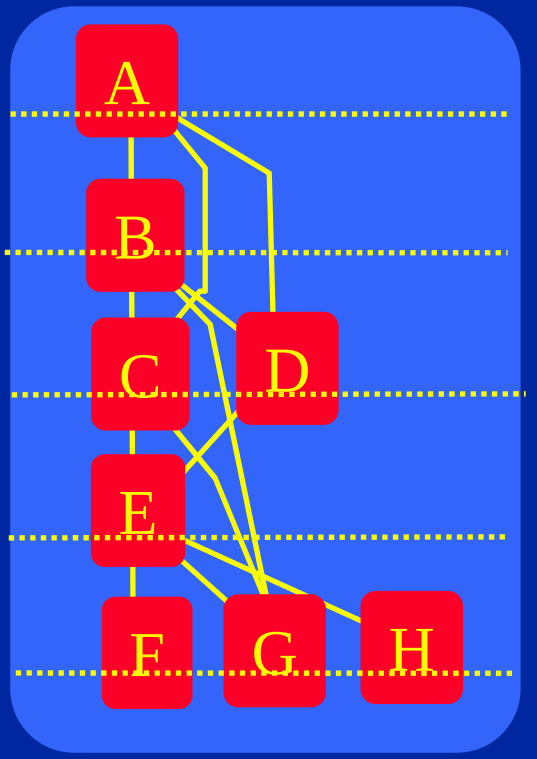
Sugiyama Step 2: Layering

Step 2

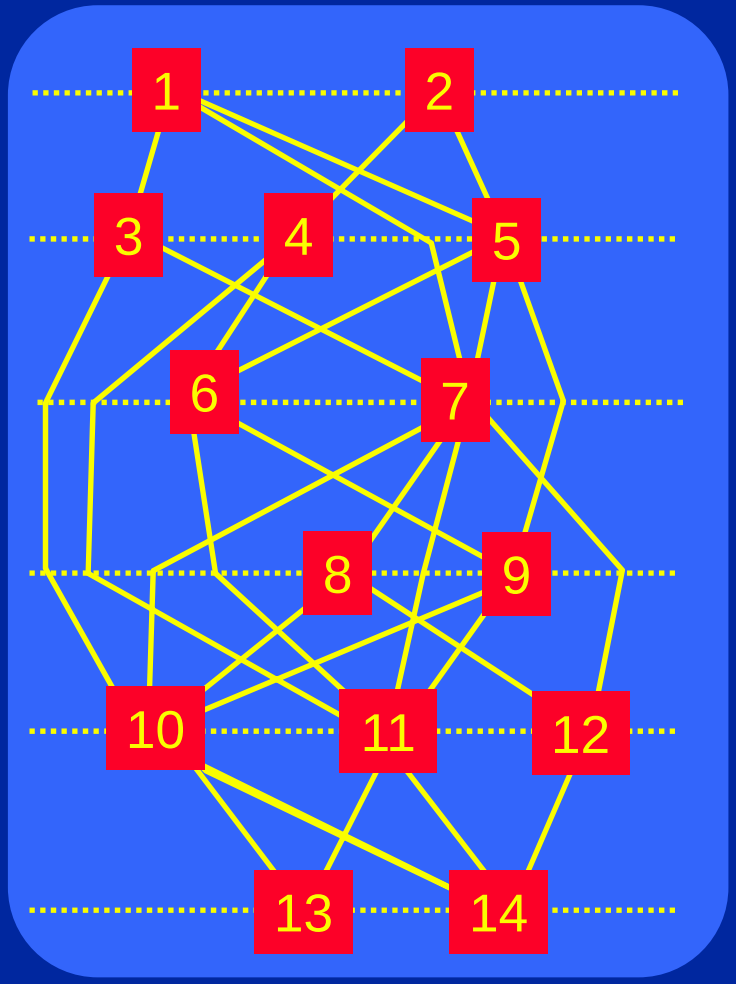
- There are several alternative layering algorithms.
- The best algorithm uses linear programming: it minimizes the total *y* extent of the edges.

A - B, C, D
B - C, D, G
C - E, G
D - E
E - F, G, H
F -
G -
H -

step 2

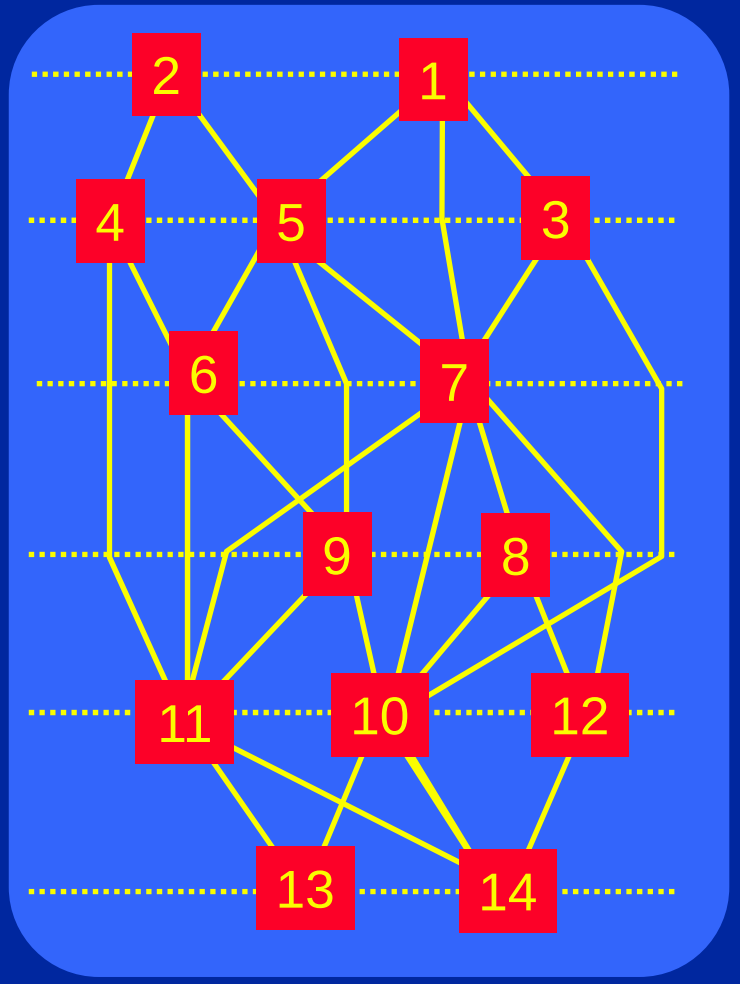


Step 3: re-arranging each layer



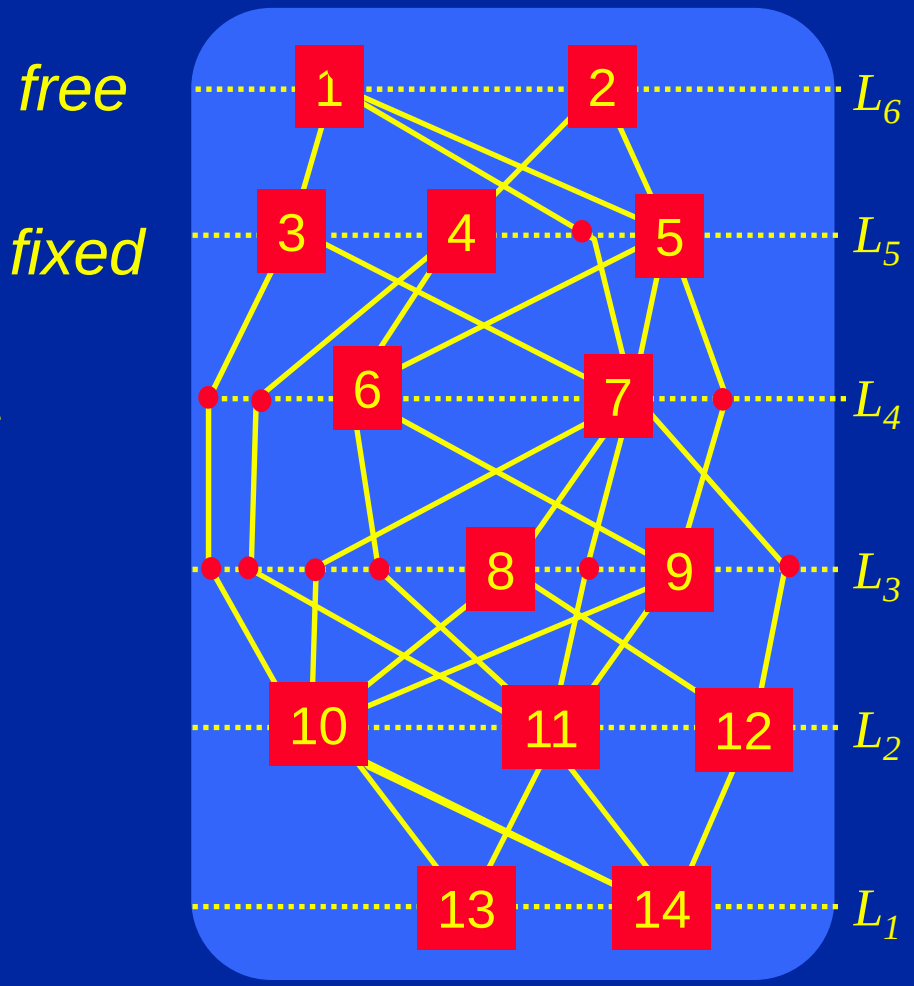
21 edge crossings

step 3



5 edge crossings

Step 3: the sweep

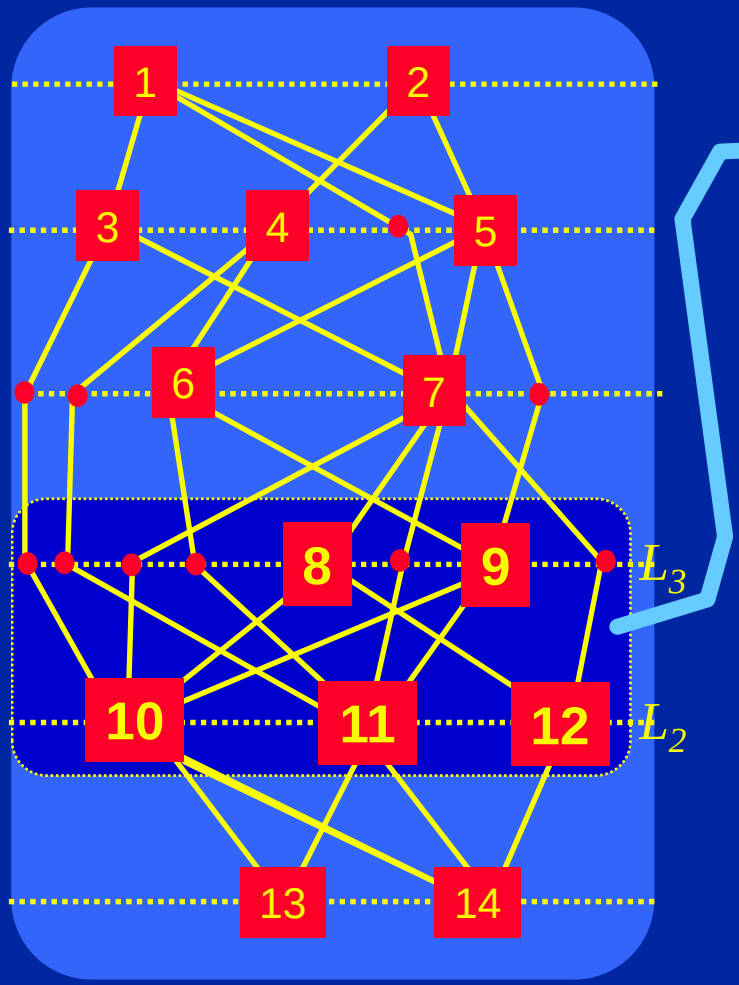


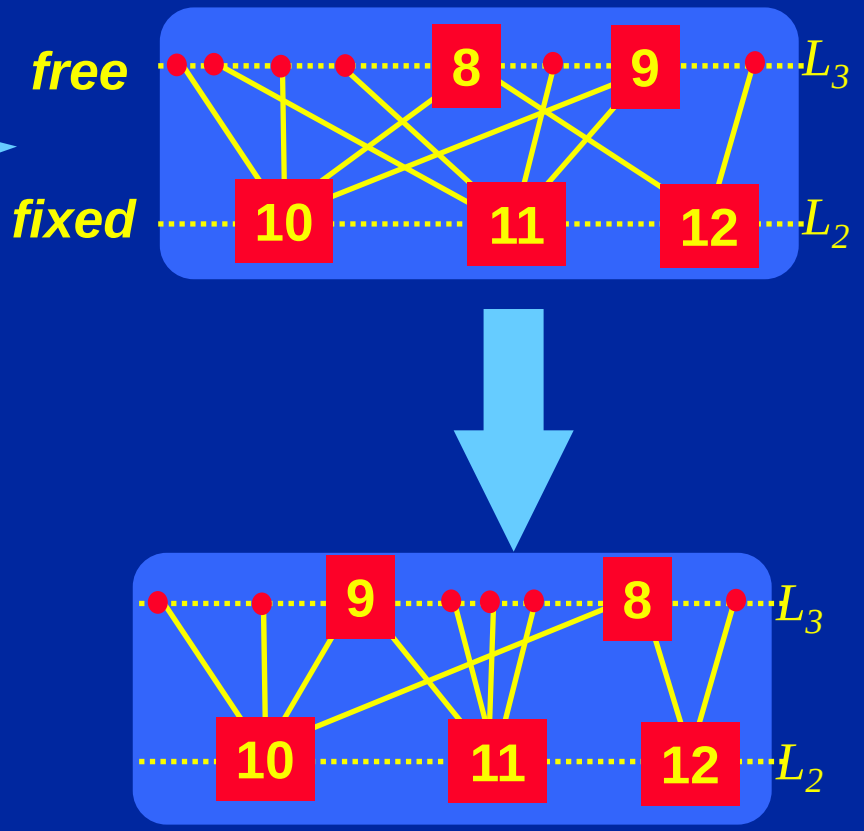
Step 3 uses a “layer-by-layer sweep”, from bottom to top.

At each stage of the sweep, we:

- hold one layer fixed, and
- re-arrange the nodes in the layer above to avoid edge crossings.

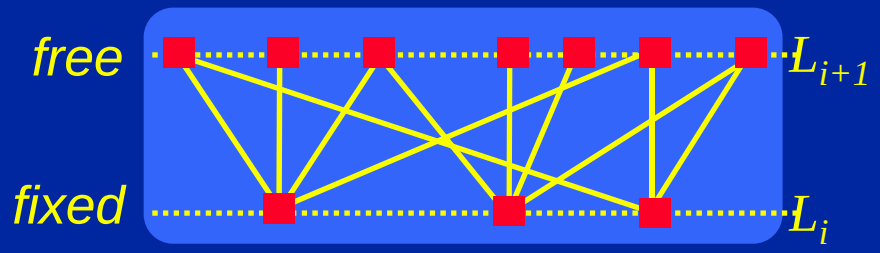
Step 3: re-arranging each layer





Step 3: re-arranging each layer

The problem of finding an optimal solution, even for two layers with one layer fixed, is NP-hard.



Heuristics

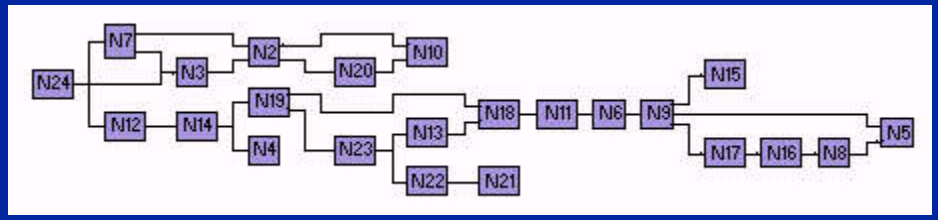
- *Barycentre algorithm*: place each free node at the barycentre (average position) of its fixed neighbors.
- *Median algorithm*: place each free node at the barycentre (average position) of its fixed neighbors.

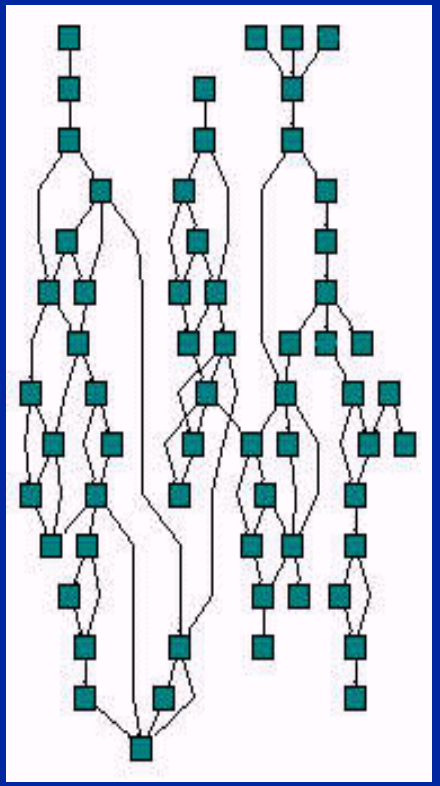
Both algorithms are efficient, and perform well in practice.

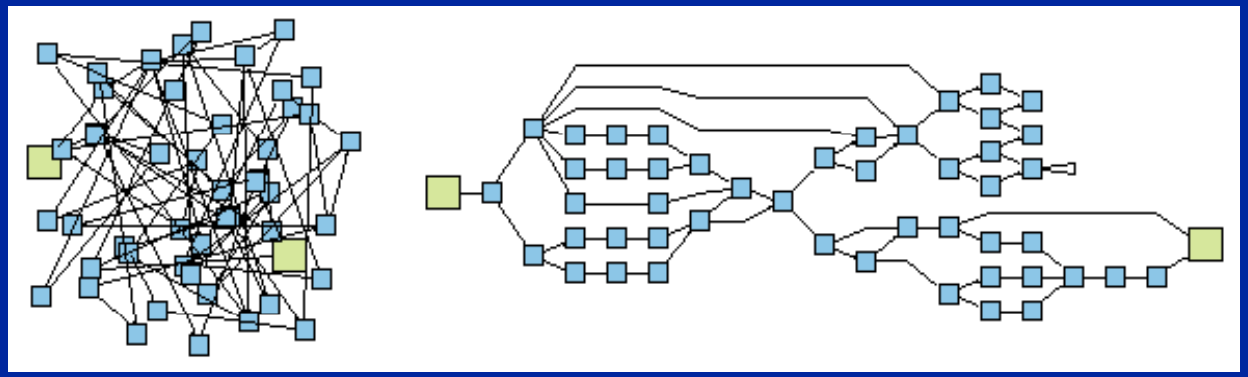
Remarks

The Sugiyama algorithm works quite well in practice.

It has been adopted in most graph drawing packages.







© Tom Sawyer Software

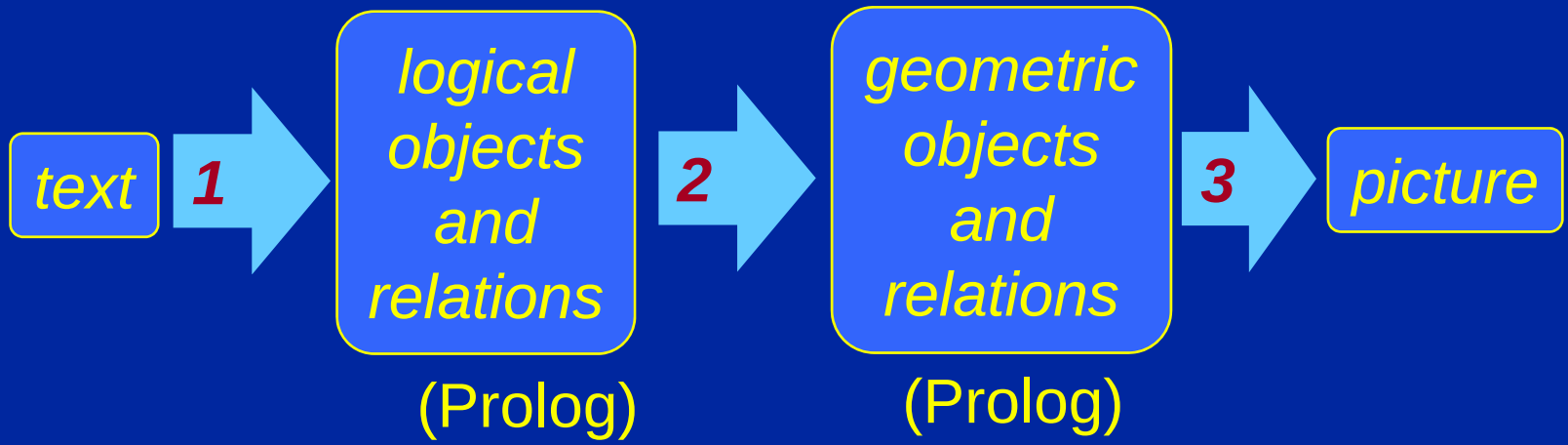
Sugiyama technique: Remarks

Sugiyama algorithm

1. readability:	*Excellent.*
2. conformance:	*Poor.*
3. controllability:	*Poor.*
4. efficiency:	*Good.*

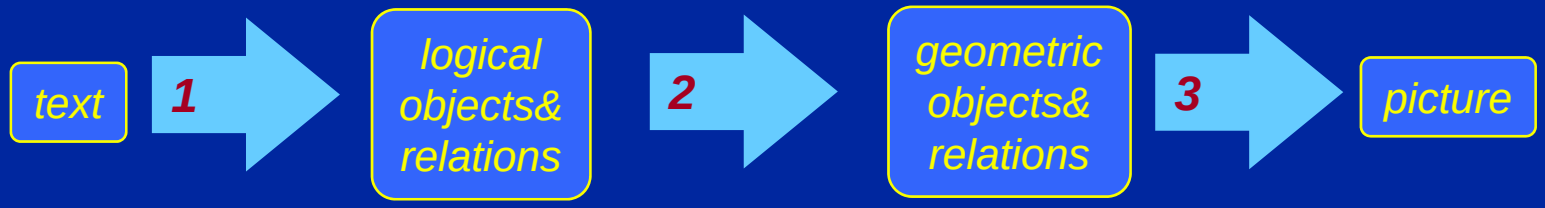
TRIP

Kamada (1989) implemented a pipeline of three steps, in an attempt to provide a general visualization system.



The system is *TRIP: TRanslation Into Pictures.*

TRIP



1. Analysis

- text -> logical objects and relations

2. Visual mapping

- logical objects -> geometric objects
- logical relations -> geometric relations
- encodes requirements of visualization in a constraint-style prolog

3. COOL

- geometric objects and relations -> picture
- constraint resolution

TRIP pipeline step 1: Analysis

Step 1: Analysis

- The text is analyzed into *logical objects and relations*.

Text

Mike is the boss of Mary, Mark and Marcia.

Mike is male, Mary and Marcia are female.

Mike, Marcia, and Mary are clerks, and Mark is the secretary.

1

Logical Objects and Relations

Boss(Mike,Mary), Boss(Mike,Mark), Boss(Mike,Marcia),

Male(Mike), Female(Mary), Male(Mark), Female(Marcia),

Clerk(Mike), Clerk(Marcia), Clerk(Mary), Secretary(Mark).

TRIP pipeline step 2: visual mapping

Step 2. The visual mapping

Step 2

Circle(X, Red) :- Male(X), Clerk(X).

Box(X, Blue) :- Male(X), Secretary(X).

Line(X, Y) :- Boss(X, Y).

Left_of(X, Y) :- Boss(Z, X), Boss(Z, Y), Clerk(X), Secretary(Y).

Above(X, Y) :- Boss(X, Y).

Line(X,Y) :- Boss(X,Y).

TRIP pipeline step 2: visual mapping

Step 2. The visual mapping: output

2

Male(Mike), Female(Mary), Male(Mark), Female(Marcia),

Clerk(Mike), Clerk(Marcia), Clerk(Mary), Secretary(Mark),

Circle(Mike, Red), Circle(Mary, Red), Box(Mark, Blue), Circle(Marcia, Red),

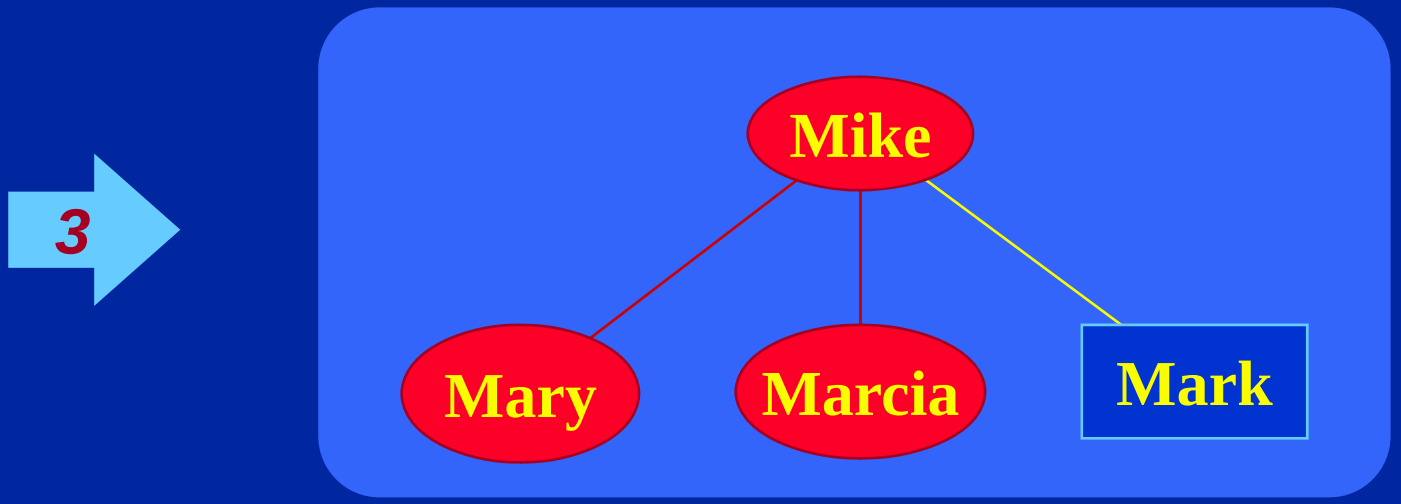
Left_of(Marcia, Mark), Left_of(Mary, Mark),

Boss(Mike, Mary), Boss(Mike, Mark), Boss(Mike, Marcia),

Line(Mike, Mary), Line(Mike, Mark), Line(Mike, Marcia).

Above(Mike, Mary), Above(Mike, Mark), Above(Mike,Marcia)

TRIP pipeline step 3: COOL



Step 3: COOL

- COOL is a constraint resolution system.
- All but the lowest level graphics primitives are handled by COOL.
- COOL handles overconstrained systems by least squares approximations.
- Rendering the output of COOL is simple.

TRIP: Remarks

The *visual mapper (Step 2)* of TRIP is easy to adjust to a specific application. It can be adjusted, for instance:

- To encode *conformance criteria* (to conform to the diagrammatic conventions of the application), for example:
  - diagrammatic conventions for family trees,
  - diagrammatic conventions for conceptual graphs.
- To allow specific users to control the drawing functions (*controllability*).

TRIP: Remarks

But TRIP drawings fail on many readability criteria

- It is difficult to express global readability goals (such as *minimizing crossings*, *minimize bends*, or *minimize area*) in TRIP.

TRIP depends crucially on constraint resolution.

- TRIP was *relatively slow* (for an interactive graphics system).
- Constraint systems sometimes have strange results.

TRIP: Remarks

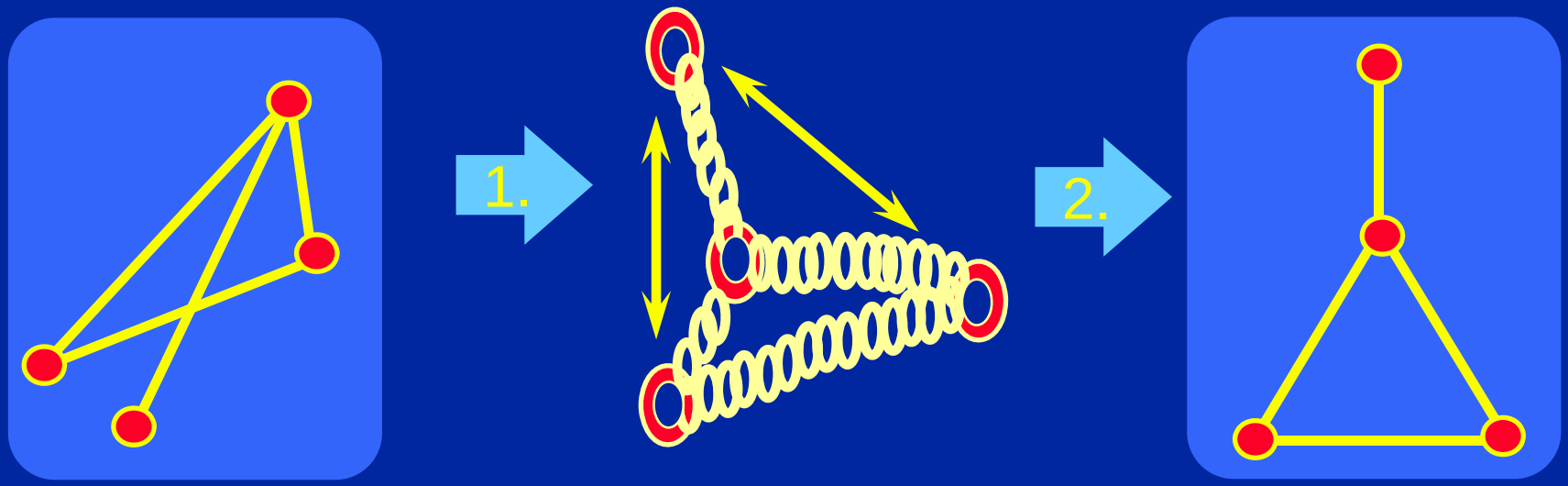
TRIP

1. *readability:* *Poor.*
2. *conformance:* *Excellent.*
3. *controllability:* *Excellent.*
4. *efficiency:* *Poor.*

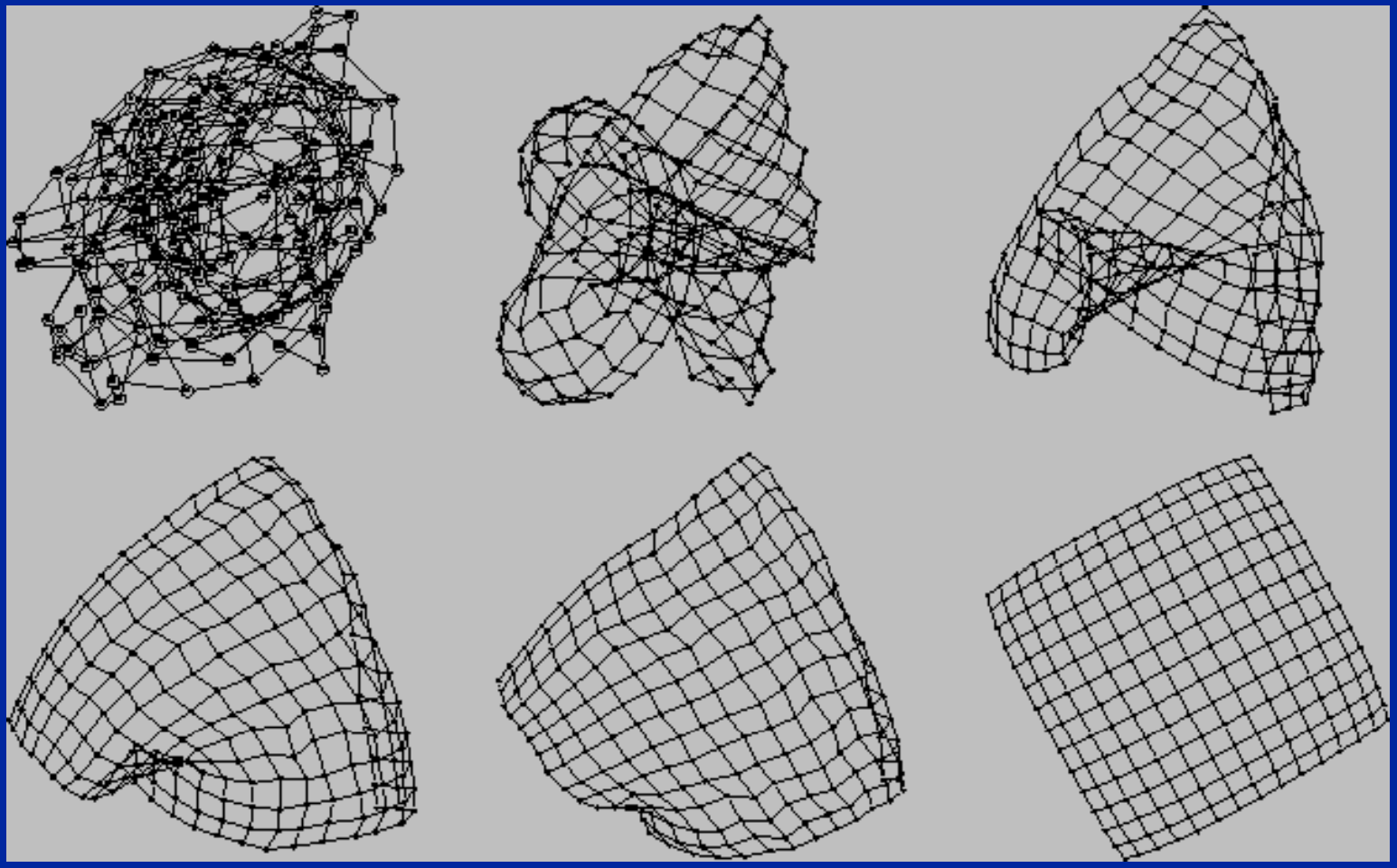
Spring techniques

Spring techniques are quite popular.

1. Place forces between pairs of nodes; for example:
   - spring forces for edges
   - gravitational repulsion for nonedges
2. Find a zero force configuration.



Spring techniques



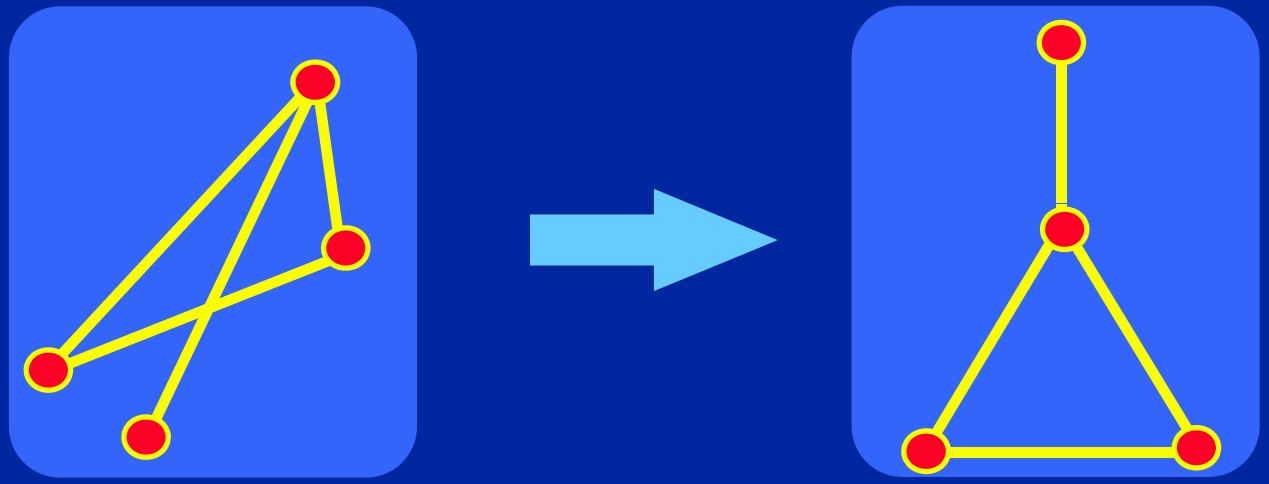
© Sander

Spring techniques

Spring algorithms can be viewed as *cost function techniques*:

1. Define a cost function which measures *ugliness*.
2. Find a drawing which minimizes the cost function.

For spring algorithms, the *ugliness* is *energy*.



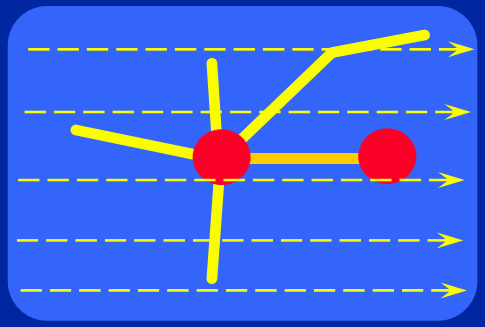
high energy (ugly)

low energy (beautiful)

Spring techniques

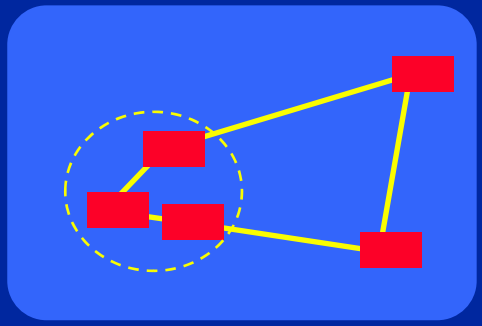
Spring techniques can be constrained in various ways.

Nails can be used to hold a node in place.



Magnetic fields and *magnetized springs* can be used to align nodes in various ways.

Attractive forces can be used to keep clusters together.



These constraints (and others) can be used for conformance and controllability.

Cost function techniques

More generally, *cost function techniques* have two parts:

1. A ***model***: a cost function which measures *ugliness*.

   $$ugliness = \lambda_1 c_1 + \lambda_2 c_2 + \ldots + \lambda_k c_k,$$

   where c_i measures one aspect of the *ugliness* of the drawing, and λ_i is a weight.

2. A ***method*** to find a drawing which minimizes the cost function, that is, minimizes *ugliness*.

Cost function techniques

The second part (the method) can be:

- a continuous optimization technique (if the cost function is continuous and smooth), or
- an Integer Linear Programming technique (if the cost function and constraints are linear), or
- a Soft Computing technique (if all else fails).

In general, there is a trade-off between generality and efficiency: the more general the model, the less efficient the method.

Cost function techniques

Tutte (1960)

- The *model* uses rubber bands and nails.
- The *method* uses Gaussian elimination.

Kamada-Kawai (about 1987)

- The *model* uses springs (and integrates with TRIP).
- The *method* is from classic Numerical Analysis, and is relatively fast.

Frick (about 1994)

- The *model* uses forces.
- The *method* uses some randomization, similar to simulated annealing. It is relatively fast.

Cost function techniques

Marks (early 1990s)

- The *model* copes with geometric constraints called *"Visual Organization Features"*.
- The *method* is a genetic algorithm.

Davidson-Harel (1992)

- The *model* uses a few forces plus edge crossings.
- The *method* is simulated annealing.

The Davidson-Harel technique showed the worst aspects of soft computing: very high computational cost for very little effect.

… two interesting techniques: *Mendonca* (1991), and *Branke-Utech* (1998)

Cost function techniques

Mendonca

- *Model*: forces plus edge crossings
- *Method*: simulated annealing

Mendonca aimed to create good drawings of ER / NIAM diagrams.

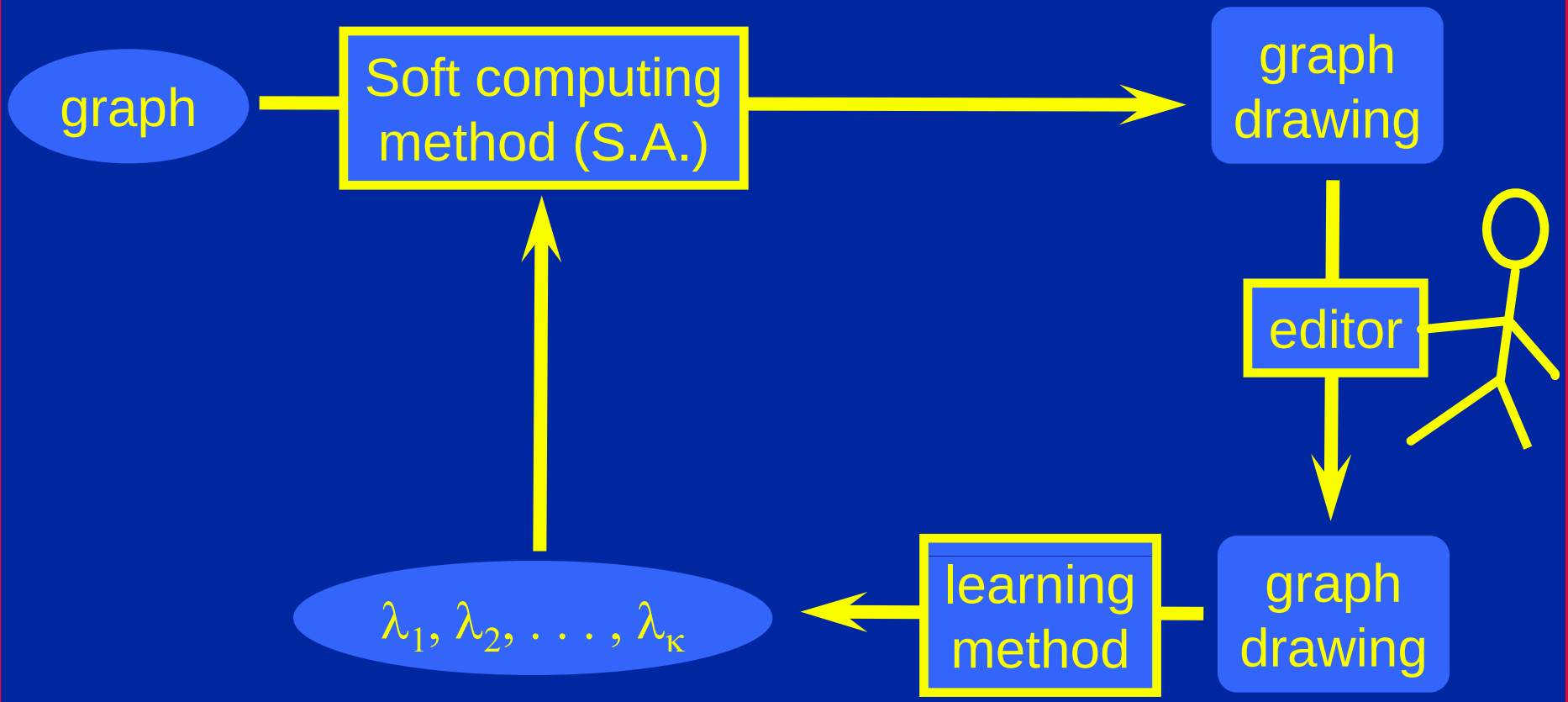
Flexibility in Mendonca's technique.

$$ugliness = \lambda_1 c_1 + \lambda_2 c_2 + \ldots + \lambda_k c_k$$

- It is easy to adjust $\lambda_1, \lambda_2, \ldots \lambda_k$ to the needs of a specific application area (conformance), or a specific user (controllability).
- Further, the system can *learn* appropriate values for $\lambda_1, \lambda_2, \ldots \lambda_k$, in a feedback control loop.

Cost function methods

Mendonca: drawing NIAM / ER diagrams



Cost function methods

Mendonca: remarks

- The methods are extremely expensive computationally.
- The number of experiments was small, but perhaps demonstrated the *feasibility* of using soft computing techniques to handle user controllability without explicit user interaction.

Cost function methods

Branke-Utech: Find a technique for drawing layered networks, better than the Sugiyama technique.

- *Model*: cost function is

  $\lambda_1 c_1 + \lambda_2 c_2$

  where

  c_1= a measure of the quality of the layering,

  c_2= number of edge crossings,

  λ_1, λ_2 are constants.

- *Method*: a genetic algorithm.

Two techniques for layered digraphs

Sugiyama technique successively solves four optimization problems:

1. *Eliminate directed cycles.*
2. *Place each node into a layer.*
3. *Order the nodes within each layer to avoid edge crossings.*
4. *Straighten the long edges.*

Branke-Utech technique

Using G.A's, the optimization problems in Steps 2 and 3 are solved together.

This gave much better (more readable) results for small graphs (less than 50 nodes).

However, the computational resources are excessive, making it impractical.

Cost function techniques: remarks

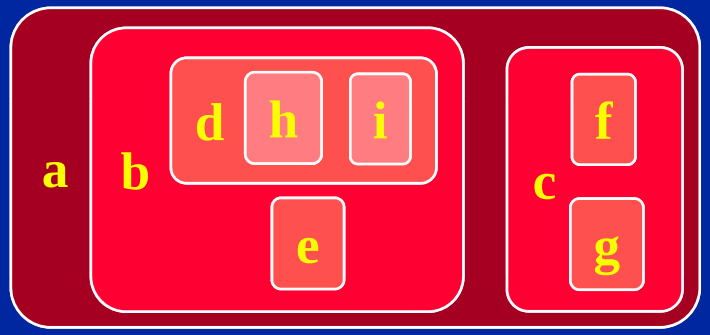
The Brake-Utech technique illustrates some advantages and disadvantages of Soft Computing:

- The flexibility of Soft Computing can give good results.
- But the computational cost can be high.

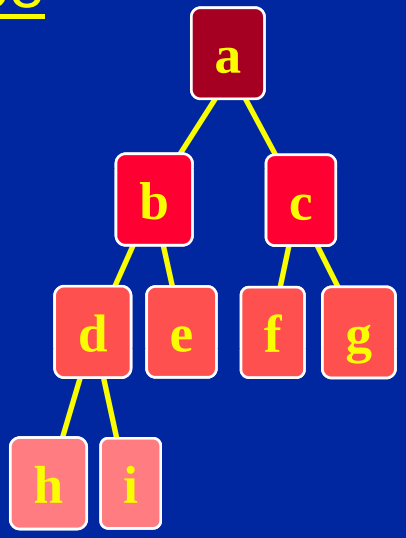
Cost function techniques: Remarks

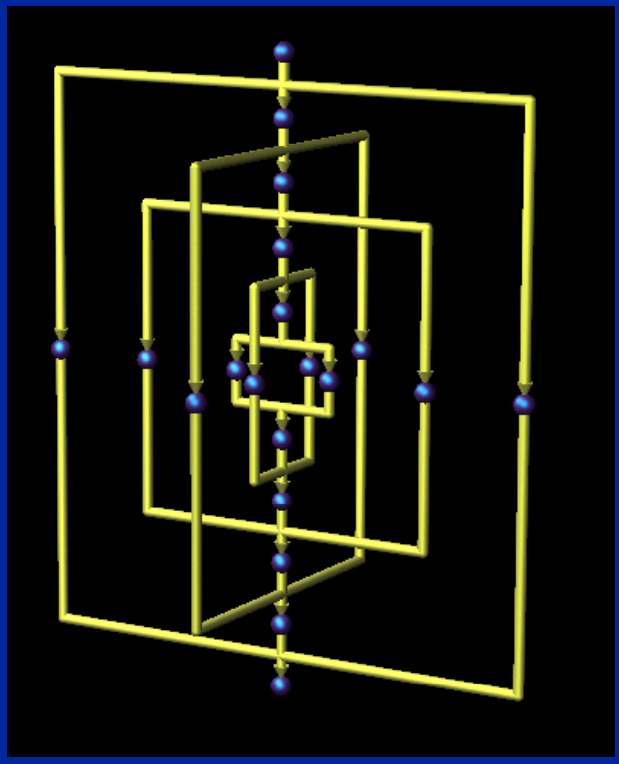
	Springs	….	Soft Computing
1. readability:	*Average*	….	*Good*
2. conformance:	*Average*	….	*Good*
3. controllability:	*Average*	….	*Good*
4. efficiency:	*Average*	….	*Poor*

Aside: Other Graph Drawing techniques



Several tree drawing algorithms

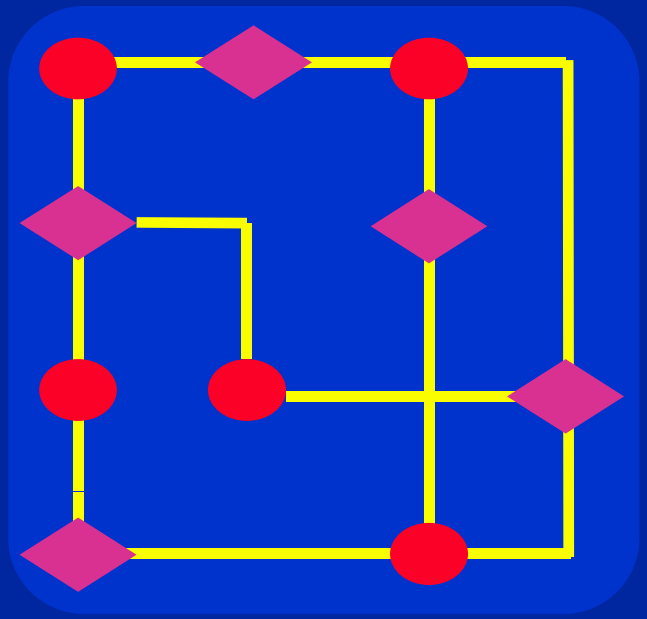


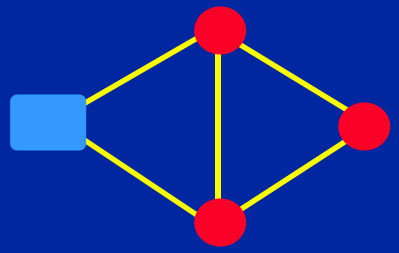


Three dimensional methods

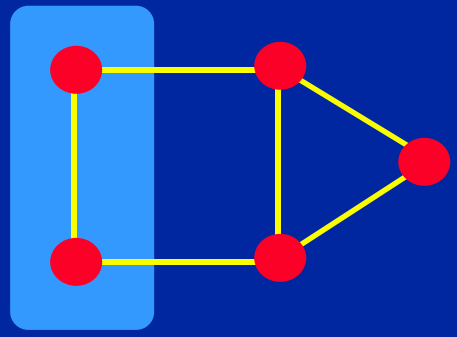
Other Graph Drawing Methods

The GIOTTO method





Graph grammar methods



Algorithmic and Declarative Approaches to Layout

There are two basic approaches to graph drawing:

- *Algorithmic Approach*: eg, Sugiyama method, GIOTTO, most tree drawing methods, visibility methods, ...
- *Declarative Approach*: eg, TRIP, graph grammar approaches, some soft computing methods, ...

Both have advantages and disadvantages . . .

Algorithmic and declarative approaches

	Sugiyama, GIOTTO	Springs	Soft Comp	TRIP
readability	****	**	***	*
conformance	*	**	***	****
controllability	*	**	***	****
efficiency	****	**	*	*

Note: Soft computing may offer a good compromise between algorithmic and declarative approaches.

Algorithmic Approach

Algorithmic Approach: *a graph drawing function is an optimization algorithm for specific optimization goals*.

Advantages

- efficient
- mathematical guarantees of effectiveness
- good for readability

Disadvantages

- fixed, hard coded optimization goals
- often bound to graph-theoretic classes (eg trees, directed graphs, planar graphs)
- often bound to specific diagrammatic conventions
- often requires specialist training

Algorithmic and Declarative Approaches to Layout

Declarative Approach: *a graph drawing function consists of a method for specifying requirements, and a generic method for obtaining a drawing according to the requirements.*

Advantages

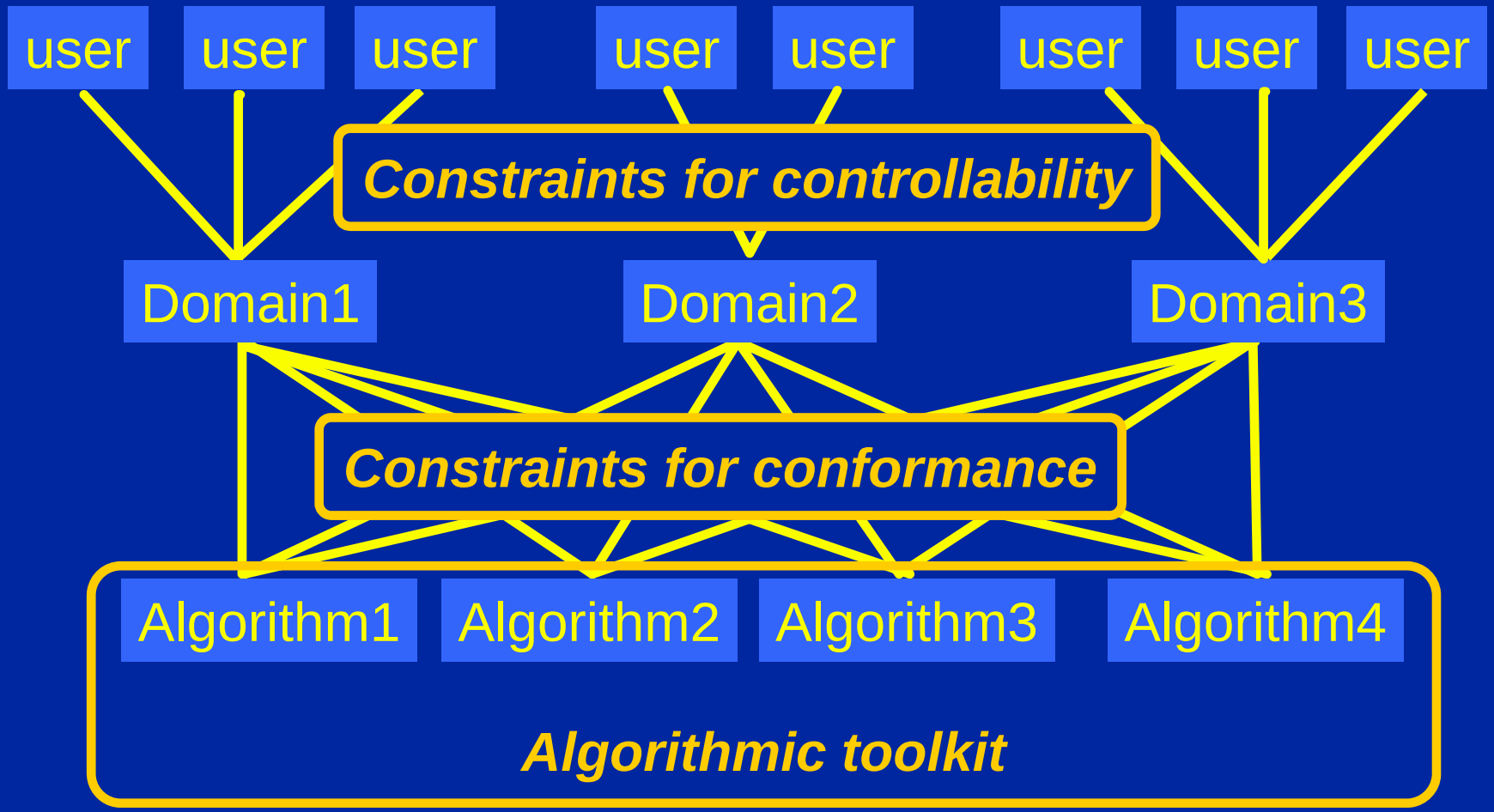
- user controllable
- conformance to different applications is easy

Disadvantages

- slow, sometimes very slow
- mathematical guarantees are mostly impossible
- some techniques perform badly with global readability goals

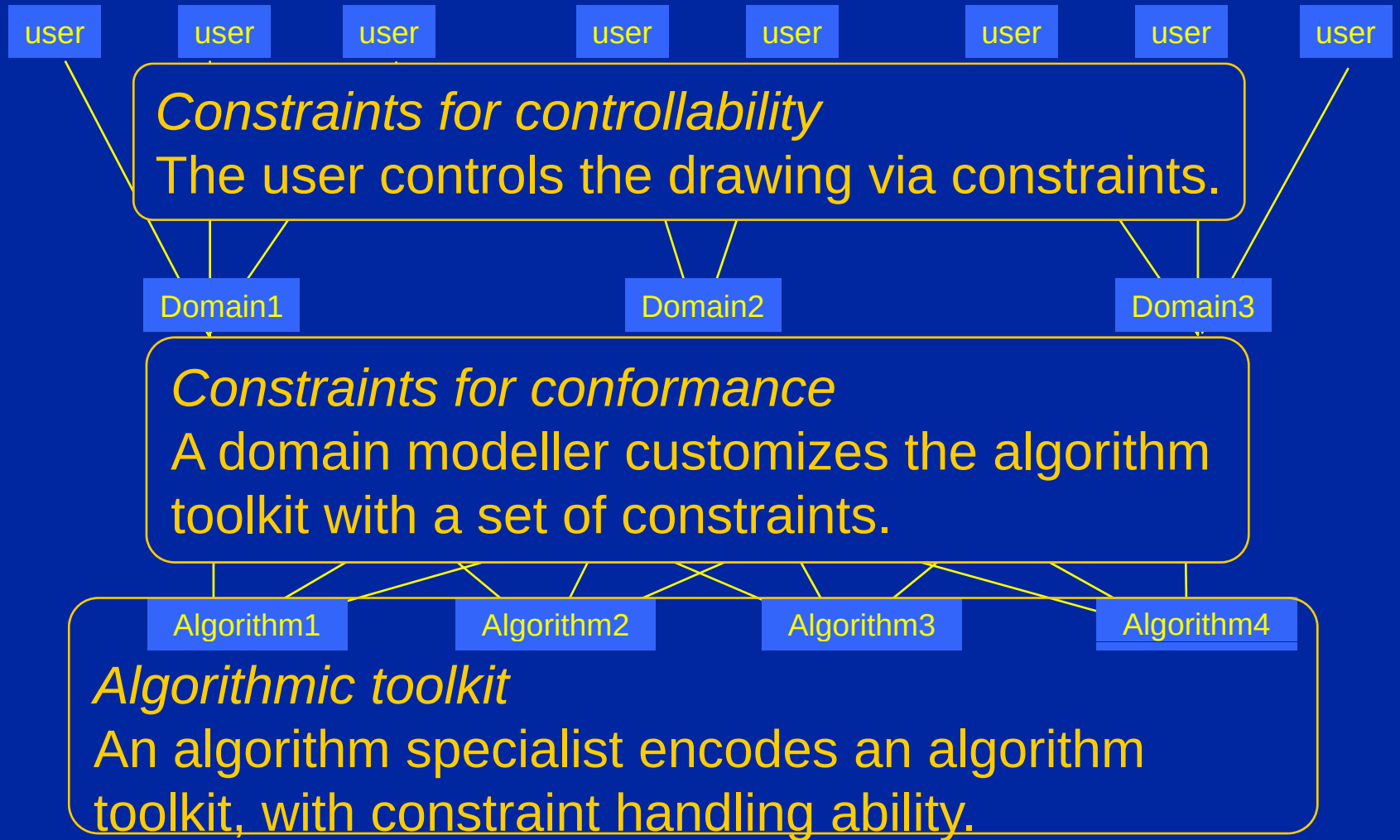
Integration

Some systems integrate algorithmic/declarative approaches, for example, *TreeSnake* (Tao Lin 1992):



Integration

TreeSnake



Algorithmic and declarative approaches

TreeSnake demonstrated a healthy integration of algorithmic and declarative techniques.

	TreeSnake
readability	****
conformance	****
controllability	****
efficiency	****

TreeSnake had many limitations, for example:

- It only covers rooted trees.
- The constrain system was very limited.

However, it did demonstrate the feasibility of integration.

Final remarks

Algorithmic and declarative approaches to Graph Drawing both have advantages and disadvantages.

- If efficiency problems can be overcome, then perhaps Soft Computing could offer a reasonable compromise between the two approaches.
- Integration of constraint handling and classical algorithmics could offer another compromise.