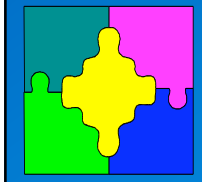
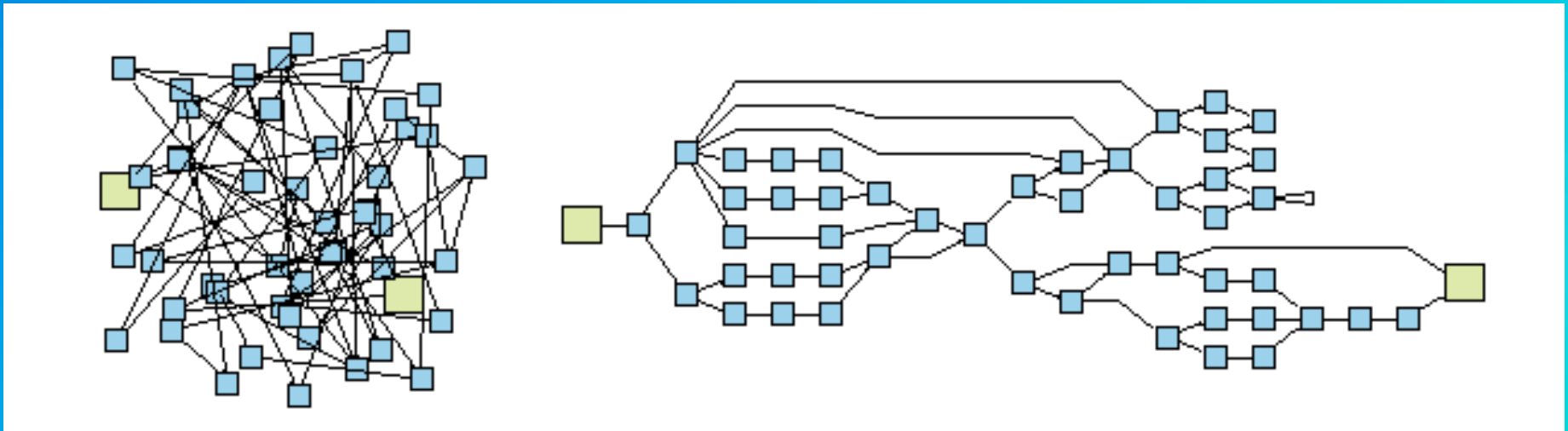
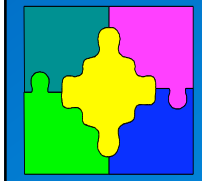


Graph Layout as Optimization



Structure can only be perceived if the layout is “readable”



Graph Drawing Applications

Graph or network-like structures are among the most commonly used types of visual representations. Automatic layout of graphs plays an important role in many applications:

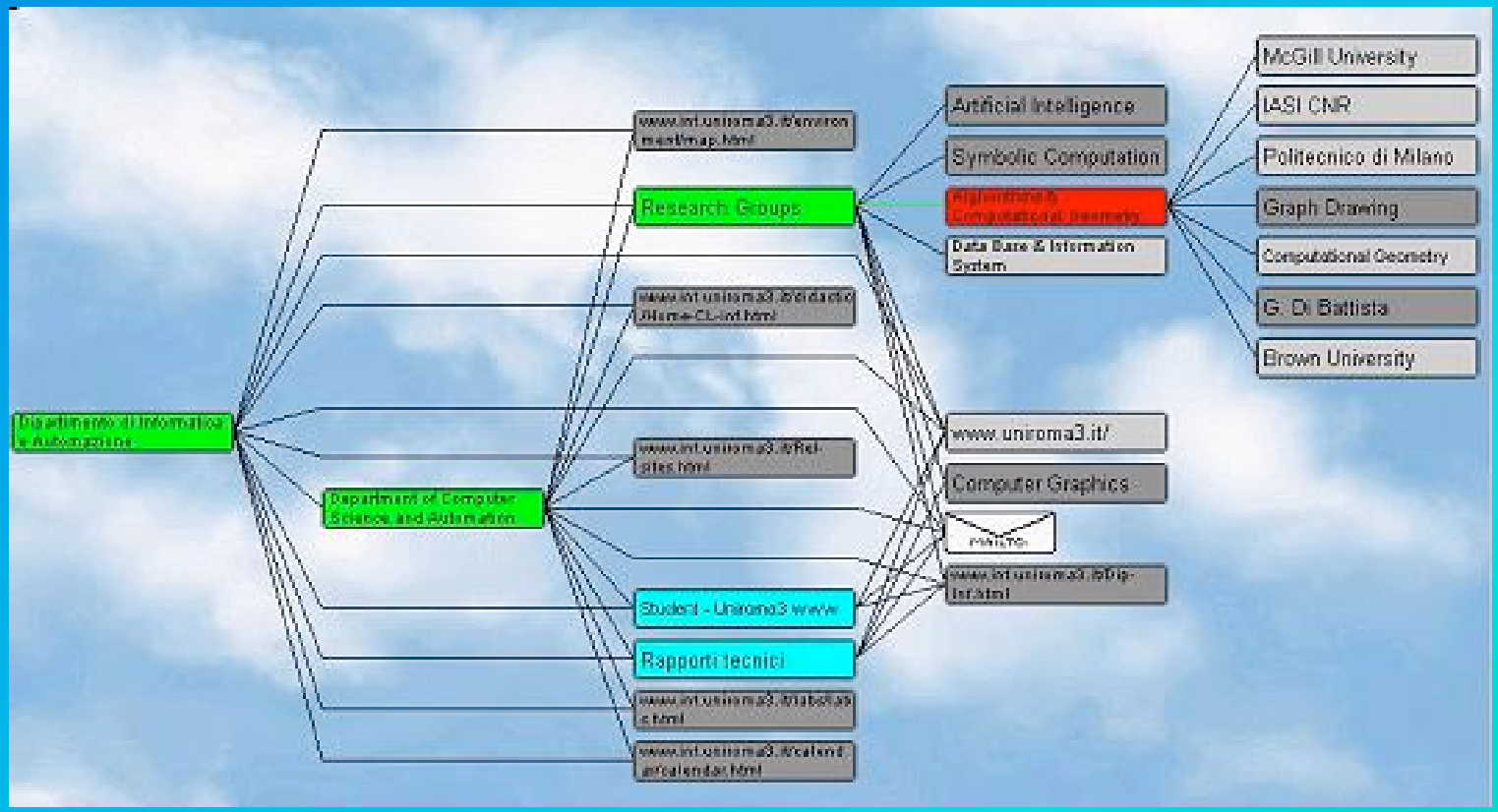
- Visual Programming
- Software Engineering
  (e.g. Flow-Charts, UML, Dependency visualization, Repository Structures)
- Engineering
  (e.g. Circuit Diagrams, Molecular Structures, Chemical Formulas)
- Sciences
  (e.g. Genome Diagrams...)

- currently a particularly hot-topic: Web-Visualization

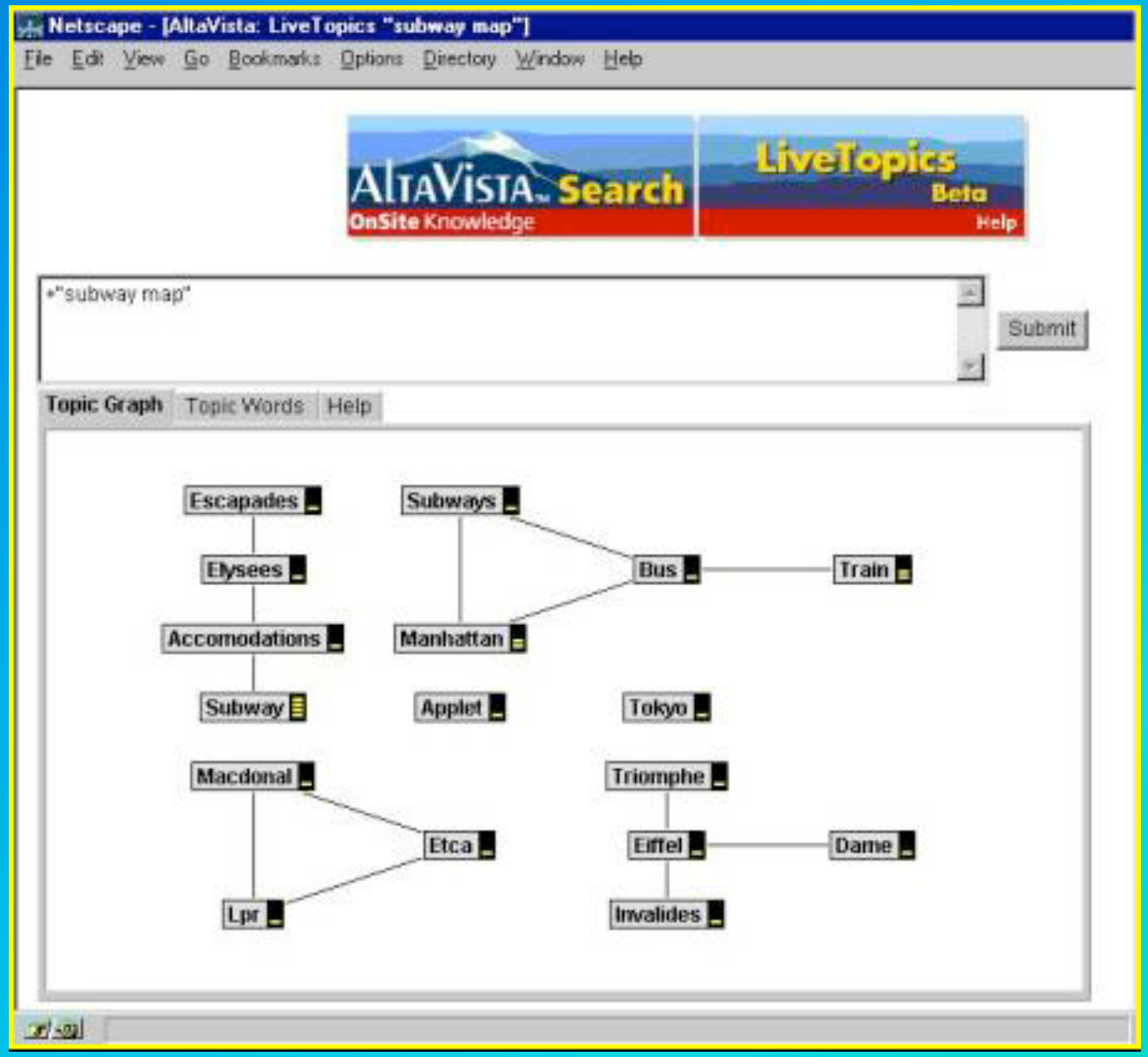
Almost always when relational data that has been obtained as the result of an automated operation such as a database or repository computation, a web-crawl or another kind of computation) has to be visualized, we are faced with the problem of automatic graph layout.

Examples of Web-Visualization

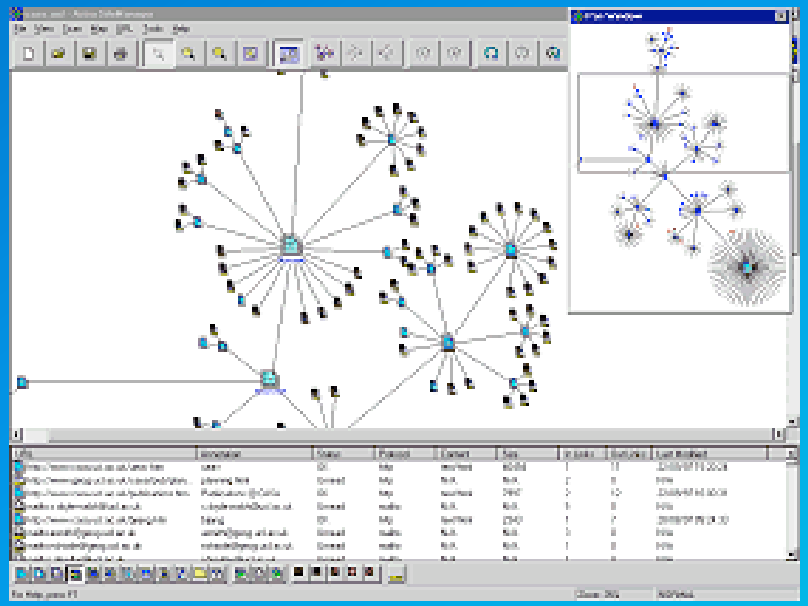
taken from the Atlas of Cyberspaces (http://www.cybergeograph.org)

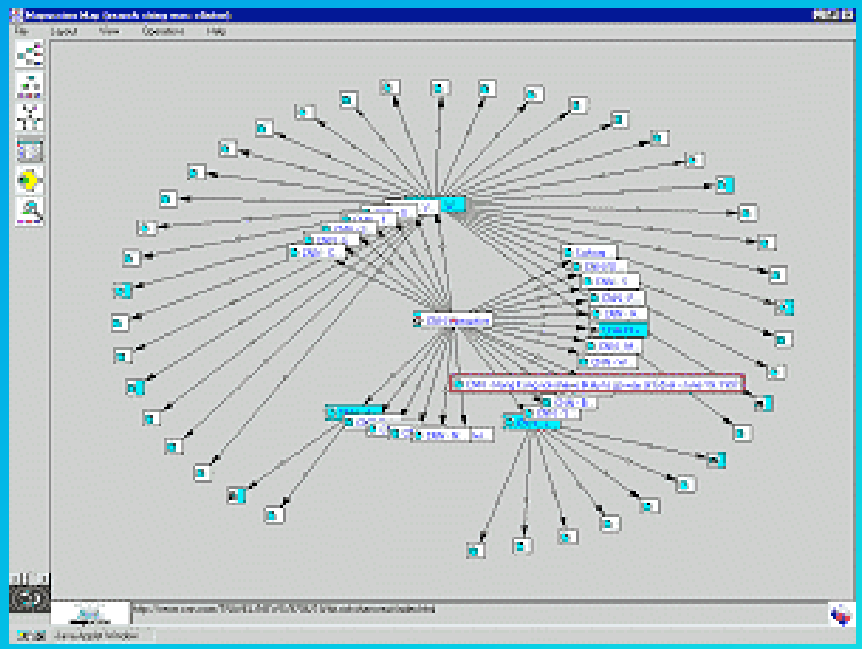


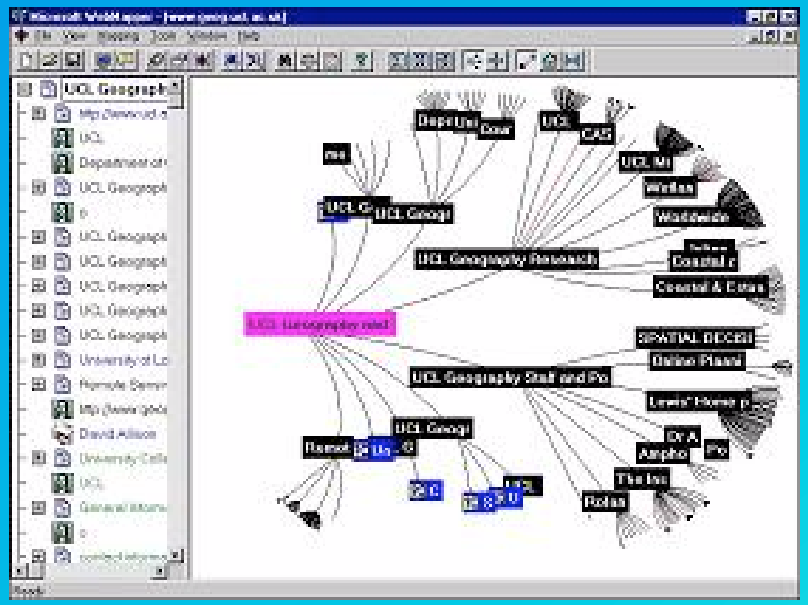
Examples of Web-Visualization

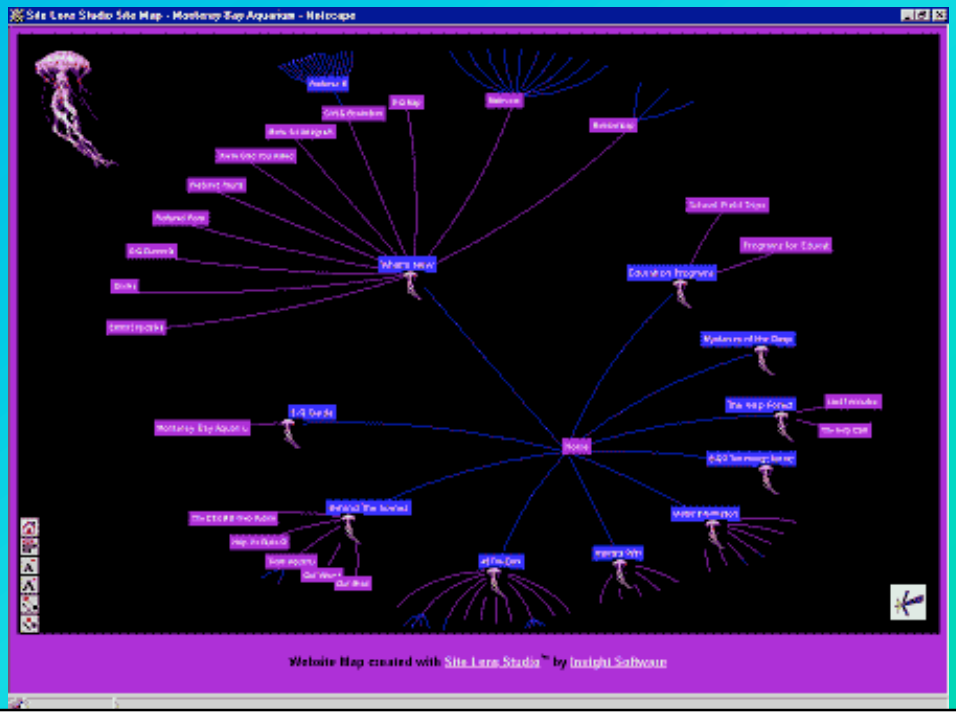


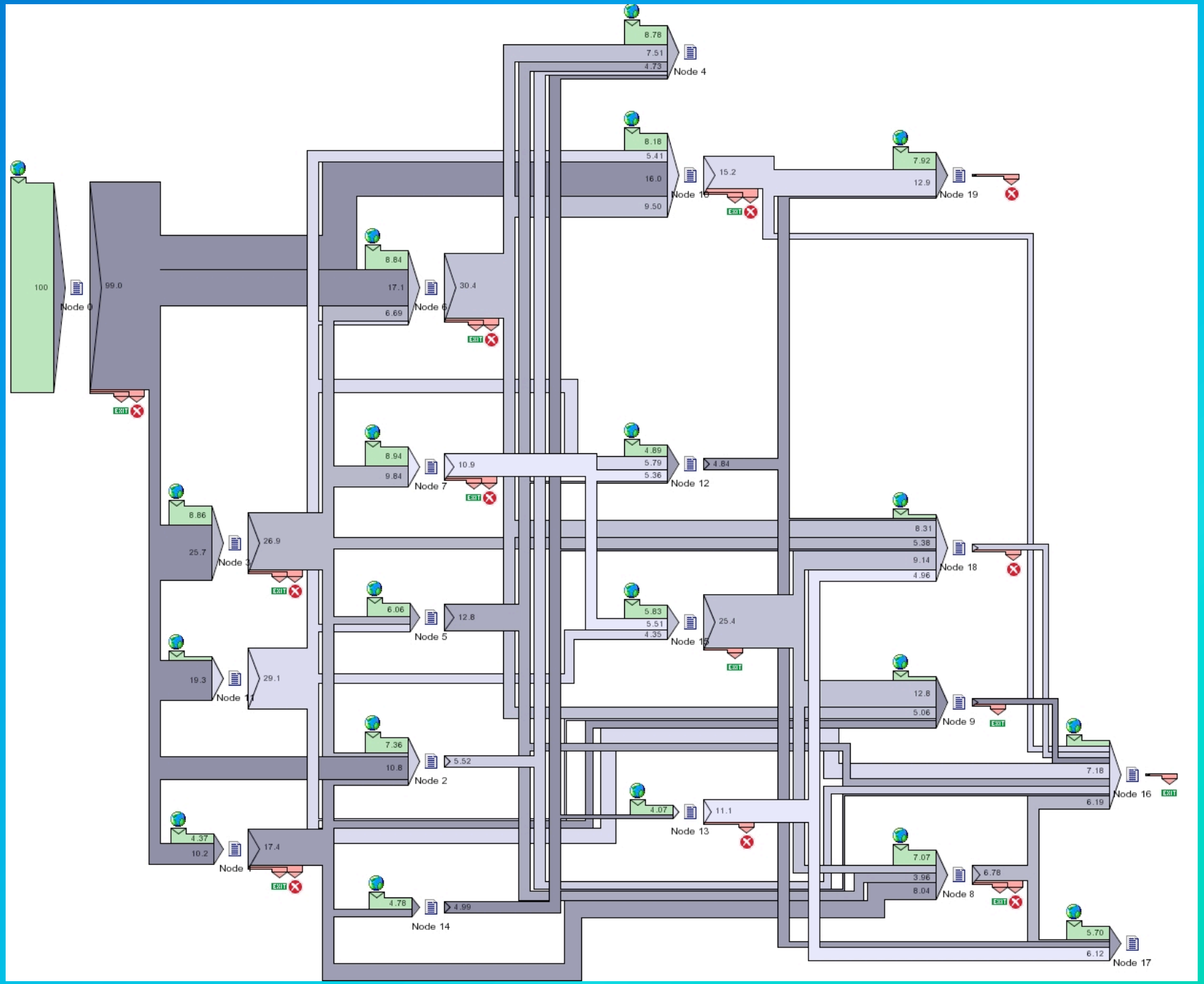
Examples of Web-Visualization

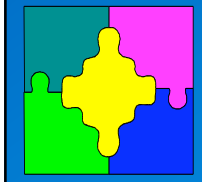









100
Node 0
99.0
EXIT
8.84
17.1
6.69
Node 6
30.4
EXIT
8.78
7.51
4.73
Node 4
8.18
5.41
16.0
9.50
Node 10
15.2
EXIT
7.92
12.9
Node 19
8.94
9.84
Node 7
10.9
EXIT
4.89
5.79
5.36
Node 12
4.84
8.86
25.7
Node 3
26.9
EXIT
8.31
5.38
9.14
4.96
Node 18
6.06
Node 5
12.8
5.83
5.51
4.35
Node 15
25.4
EXIT
19.3
Node 11
29.1
12.8
5.06
Node 9
EXIT
7.36
10.8
Node 2
5.52
7.18
6.19
Node 16
EXIT
4.07
Node 13
11.1
4.37
10.2
Node
17.4
EXIT
7.07
3.96
8.04
Node 8
6.78
EXIT
4.78
Node 14
4.99
5.70
6.12
Node 17



Graph Drawing as a Research Field

Automatic layout of graphs is a very complex and mathematically challenging problem. It has therefore spawned a whole field of research with its own conference series: The International Symposium on Graph Drawing (since 1992)

Proceedings are published in Springer-Verlag's
"Lecturer Notes in Computer Science series"

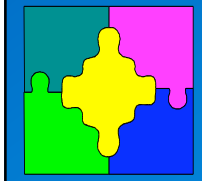
The best (and only comprehensive) book on the topic is:
Graph Drawing by G. DiBattista, P. Eades, R. Tamassia and I.G. Tollis,
Prentice Hall, 1999.

All references in these notes are listed in this book's bibliography.

One of the authors also maintains a very good web page on this topic:
http://www.cs.brown.edu/people/rt/gd.html

From here you can also reach the pages for the symposium.

Another interesting book is: *Graph Drawing Software*
by Michael Jünger and Petra Mutzel. Springer Verlag, Berlin, 2004.



Different Types of Graphs

Many different kinds of graphs have to be investigated.
Most commonly we use:

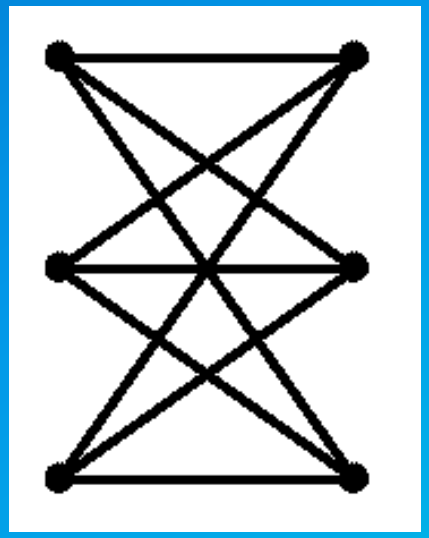
- Trees (binary, ordered, rooted, free, ...)
- Directed Acyclic Graphs
- General Graphs (directed, undirected, ...)

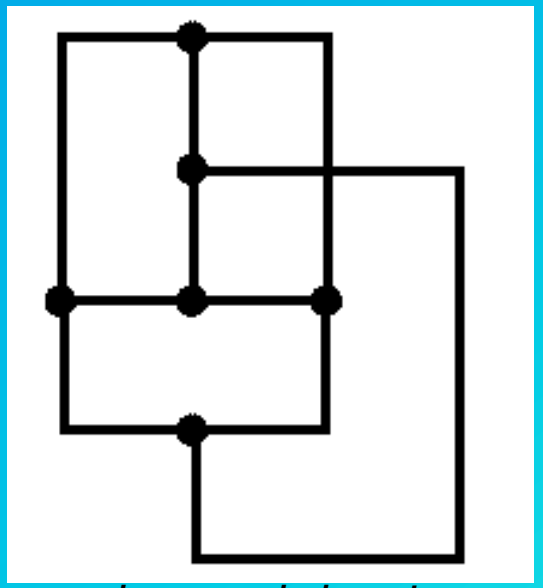
For each of these we can adopt different kind of drawing conventions.
For example

- Trees can be drawn rooted or free
- Trees can be drawn “normal” or radial, hyperbolic

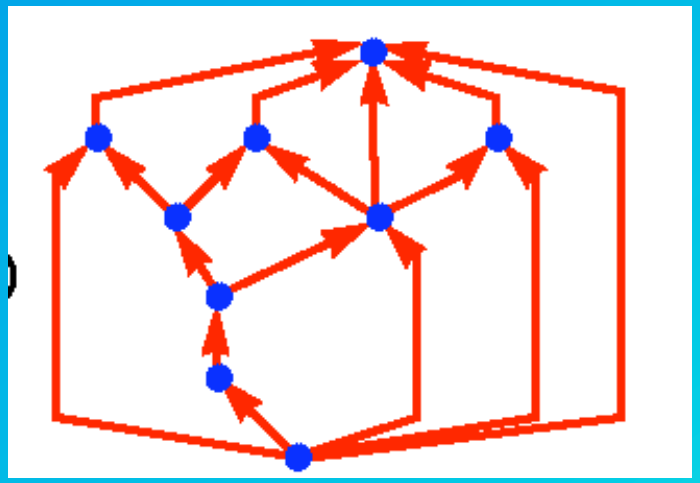
- Any graph can be drawn as a
  straight-line drawing
  orthogonal drawing

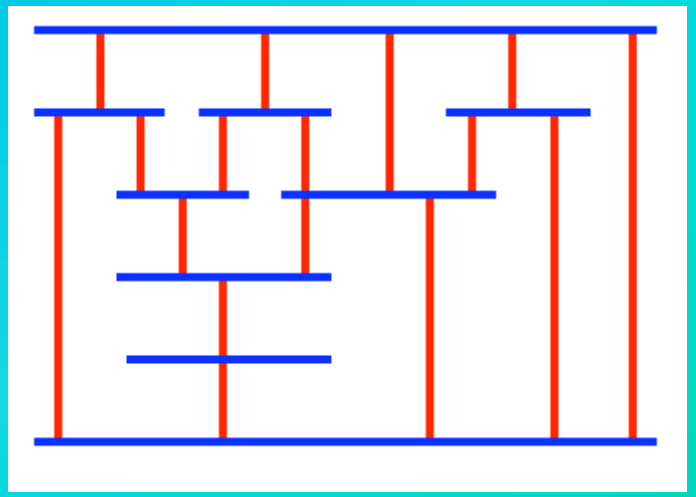
Orthogonal Drawings



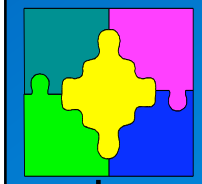


orthogonal drawing



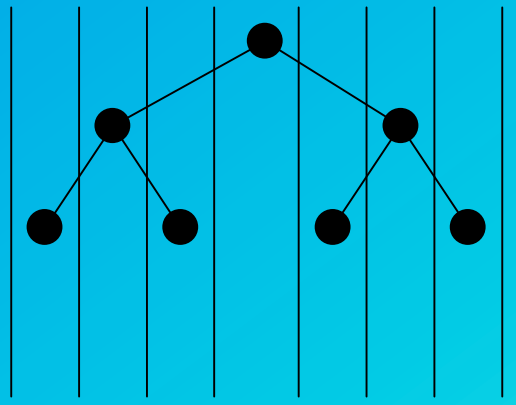


visibility drawing



A Trivial Layout Algorithm

in special cases a layout can be computed directly.

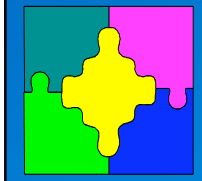


Example: Trivial tree drawing algorithm

```
class tree { left: tree; right:tree;
             posX:int; posY: int;
             isLeaf: boolan;

 int calcPos(int leftOf, int depth) {
          if (! this.isLeaf()) {
                    this.posX=left.calcPos(leftOf, depth+1);
                    this.posY=depth;
                    return right.calcPos(posX+1, depth+1);
          }
          else {    this.posY=depth;
                    return leftof;
          }
 }
 }
```

called with root.calPos(0,0) this method calculates the coordinates for all nodes in a tree in a (non-dense) standard layout.

However, if we want to optimize the drawing for a particular aspect this becomes usually impossible. Optimization of almost any interesting property is NP-hard (e.g. number of edge crossings).



Graph Drawing Aesthetics

In general, we want to optimize the drawing for
comprehensibility and aesthetics.

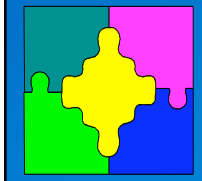
But what does this mean (mathematically)?

The most commonly used aesthetic criteria are:

- expose symmetries
- make drawing compact / fill available space

It turns out that many “higher-level” aesthetic criteria are implicit consequences of:

- minimized number of edge crossings
- evenly distributed edge length
- evenly distributed node positions
- sufficiently large node-edge distances
- sufficiently large angular resolution

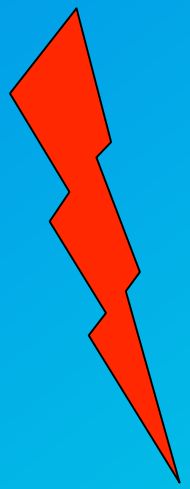


Graph Drawing as Optimization

The aesthetic objective function takes node positions as input and returns a numerical measurement of its “beauty”.

f(g) = c1 * #crossings(g) +
c2 * 1/std-deviation-edge-length(g) +
...

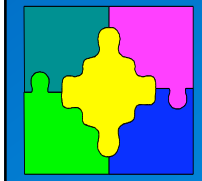
This turns graph layout into a general optimization problem



Most of the sub-problems, e.g. the mimimization of edge-crossings are already known to be *NP*-hard.

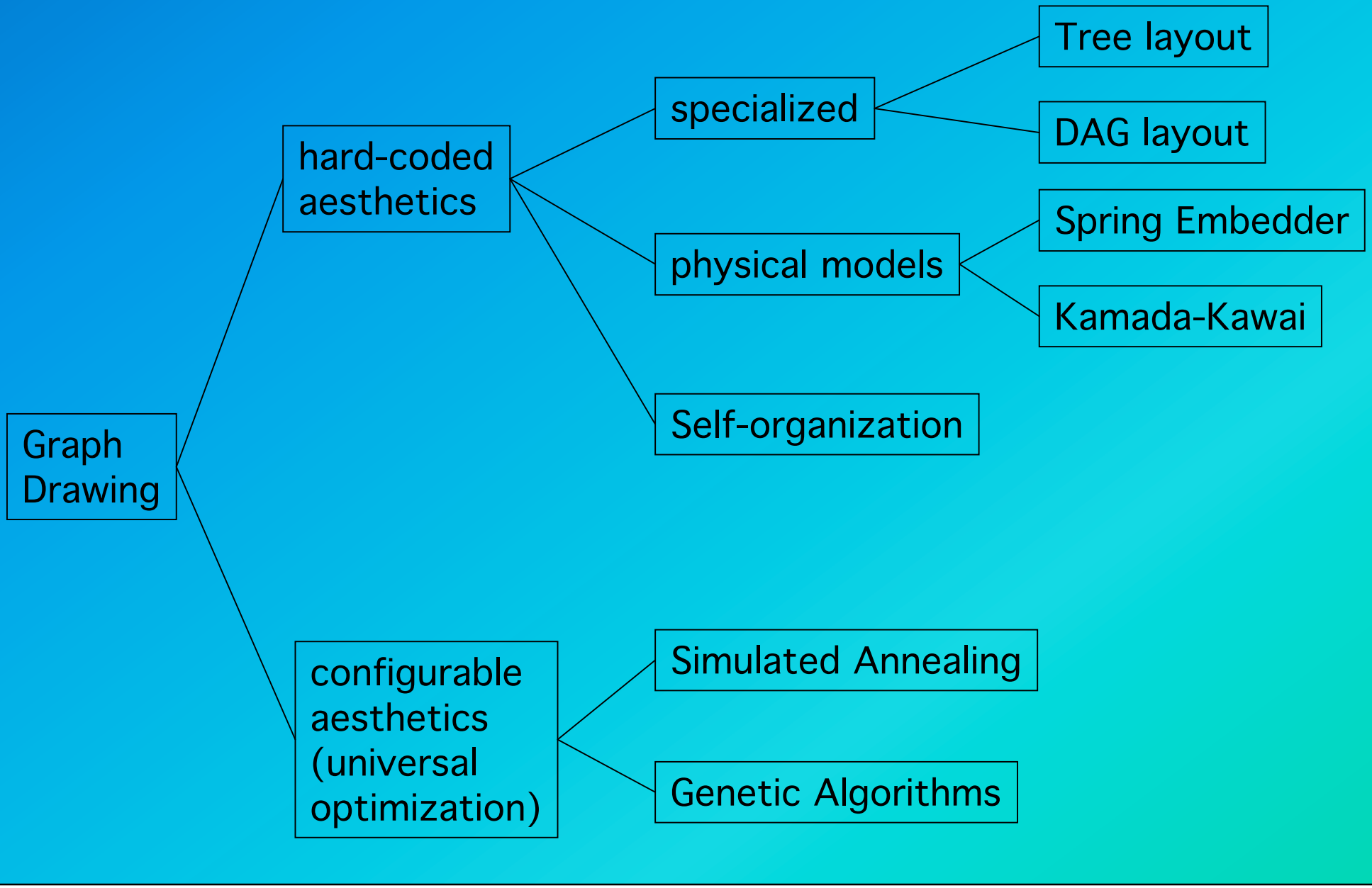
A straight-forward full optimization is often not feasible.

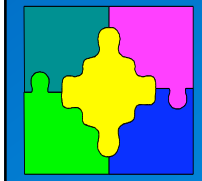
Usually stochastic / heuristic methods are employed.



Approaches to Automatic Layout

(an incomplete overview)

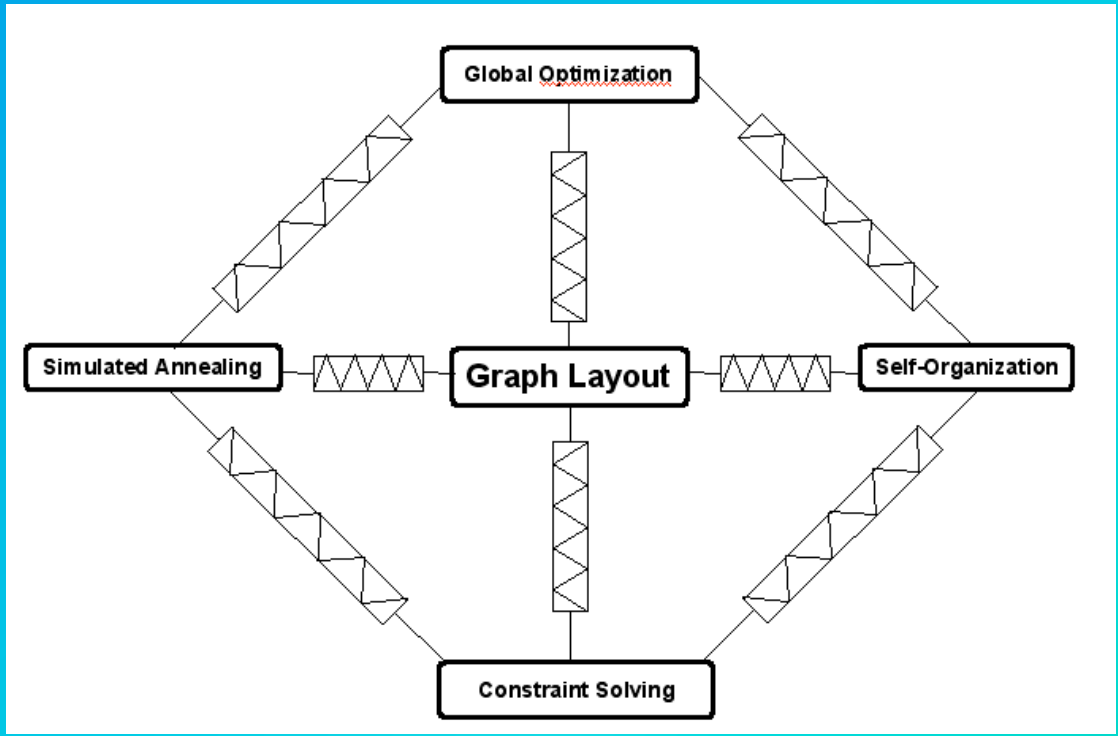




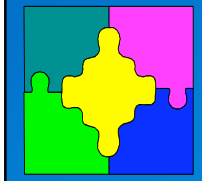
Spring Embedder Layout (Idea)

The general idea of spring-embedder or force-directed layout is to work on a physical model of the graph in which the nodes are represented by steel rings and the edges are springs attached to these rings.

If such a “physical graph” is allowed to move without restrictions, it will move to a configuration where the potential energy in the springs is minimized.



A spring-embedder layout method emulates this process algorithmically.



Energy Analysis

According to Hooke's law (spring law), the spring force is approximated by

$$F_s = -k_s\left(len - len_0\right) = -k\Delta len$$

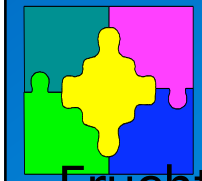
where len0 is the natural length of the (resting) spring.

We also add a second, repulsive force, which is used to achieve even node distribution. This force is modelled as an analogy to electrical forces.

$$F_e = \frac{k_e}{d^2}$$

Attractive forces are only considered between incident nodes, repulsive forces between all nodes. Thus the total force acting at a node is:

$$F = \sum_{(u,v)\in E} k_s(d(u,v) - l) + \sum_{(u,v)\in V\times V} \frac{k_e}{d(u,v)^2}$$



Fruchterman-Rheingold (Idea)

Fruchterman and Rheingold [FR91] have defined as simple heuristic approach to force-directed layout that works surprisingly well in practice.

The basic idea is to just calculate the attractive and repulsive forces at each node independently and to update all nodes iteratively.

The forces take a somewhat different form (justified by experimentation).
The attractive force is:

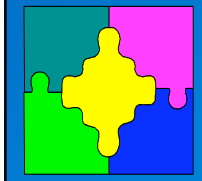
$$f_a(x) = \frac{x^2}{k}$$

where k is chosen as $k = \sqrt{\frac{area}{|V|}}$

and the repulsive force is: $f_r(x) = \frac{k^2}{x}$

Additionally, the maximum displacement of each node in an iteration is limited by a constant that is slightly decreased with each iteration.

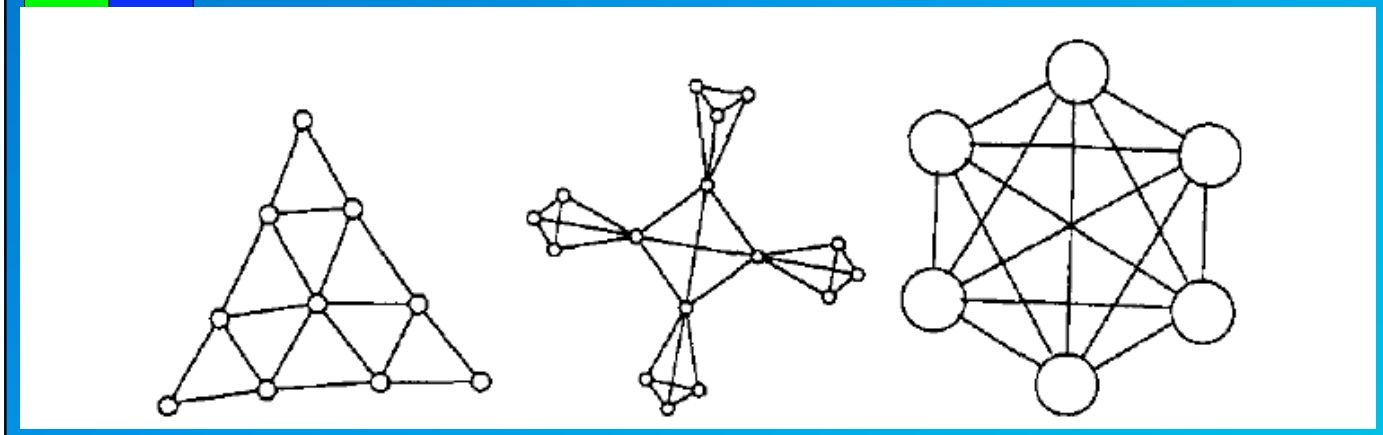
A very compact description is given in: “Simulating Graphs as Physical Systems”, A. Frick, G. Sander and K. Wang in Dr. Dobbs Journal, August 1999.

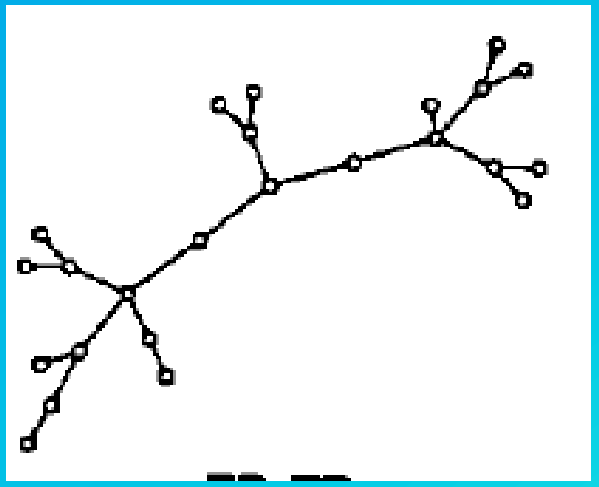


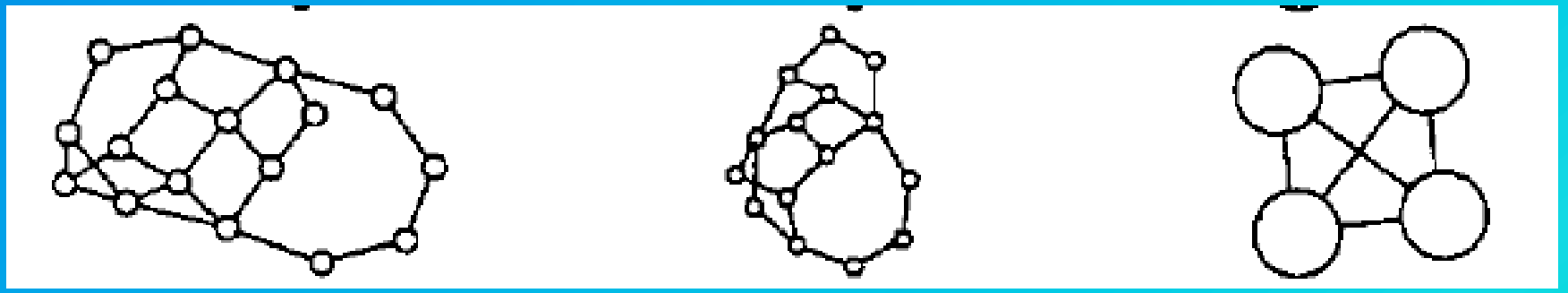
Fruchterman-Rheingold (Algorithm)

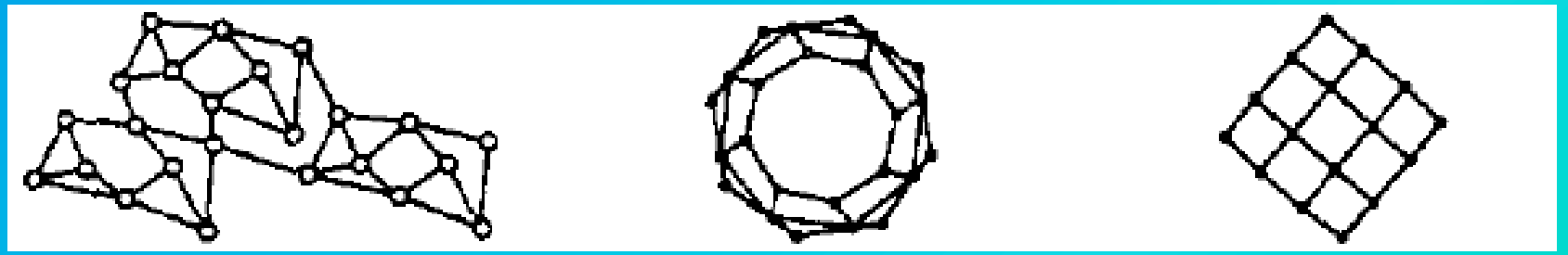
```
for i := 1 to max_iterations do begin
    for each v in Vertices do begin { calculate repulsive forces}
        v.pos' := (0,0)
        for each u in Vertices do
            if (u != v) then begin
                Delta := v.pos - u.pos;
                v.pos' := v.pos' + (Delta / len(Delta) ) * f_r(len(Delta))
            end
        end

    for each e in Edges do begin { calculate attractive forces }
        Delta := e.start.pos' - e.end.pos';
        e.start.pos' := e.start.pos' + (Delta / len(Delta) ) * f_a(len(Delta));
        e.end.pos'   := e.end.pos' + (Delta / len(Delta) ) * f_a(len(Delta));
    end

    for v in Vertices do begin { limit displacement }
        Delta := v.pos' - v.pos;
        v.pos := v.pos + (Delta / len(Delta) ) * min(len(Delta), t);
    end

    reduce temperature t;

end
```

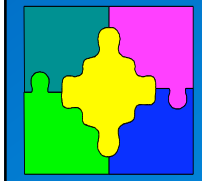
Fruchterman-Rheingold (Examples)











Kamada-Kawai Layout

Another very popular algorithm for force-directed layout is Kamada-Kawai (KK)

See: T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. Information Processing Letters 31(1989):7-15.

It is also based on the idea of a balanced spring system and energy minimization.

However, in difference to FR-Layout, KK-Layout attempts to utilize the derivatives of the force equations to achieve faster convergence.

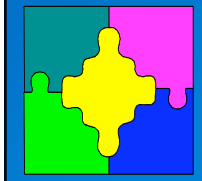
In KK-Layout the global energy is defined as:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \tfrac{1}{2} k_{i,j} \left(\left| p_i - p_j \right| - l_{ij} \right)^2$$

where

p_k is the position of vertex k

$l_{ij} = c \cdot d_{ij}$ is proportional to the topological distance d_{ij} of vertex i and j



Kamada-Kawai Energy Minimization

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \tfrac{1}{2} k_{i,j} \left(\left| p_i - p_j \right| - l_{ij} \right)^2 \quad \text{can be rewritten as}$$

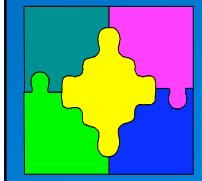
$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \tfrac{1}{2} k_{i,j} \left[\left(x_i - x_j \right)^2 + \left(y_i - y_j \right)^2 + l_{ij}^2 - 2 \cdot l_{ij} \cdot \sqrt{\left(x_i - x_j \right)^2 + \left(y_i - y_j \right)^2} \right]$$

For a energy minimum we must have

$$\forall_j \; \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} = 0 \quad \text{with}$$

$$\frac{\partial E}{\partial x_j} = \sum_{i \neq m} k_{im} \left[\left(x_m - x_i \right) - \frac{l_{mi} \cdot \left(x_m - x_i \right)}{\sqrt{\left(x_m - x_i \right)^2 + \left(y_m - y_i \right)^2}} \right] \qquad \text{(Eq. 1)}$$

$$\frac{\partial E}{\partial y_j} = \sum_{i \neq m} k_{im} \left[\left(y_m - y_i \right) - \frac{l_{mi} \cdot \left(y_m - y_i \right)}{\sqrt{\left(x_m - x_i \right)^2 + \left(y_m - y_i \right)^2}} \right] \qquad \text{(Eq. 2)}$$

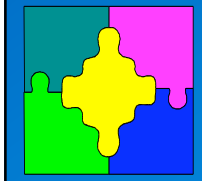


Kamada-Kawai Energy Minimization

Unfortunately, these equations are not independent.
Therefore they cannot independently be brought to zero.

KK-Layout employs a heuristic in which only a single vertex is moved at a time. This is chosen to be the “most promising” vertex with the maximum gradient value of

$$\Delta_m = \sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2}$$



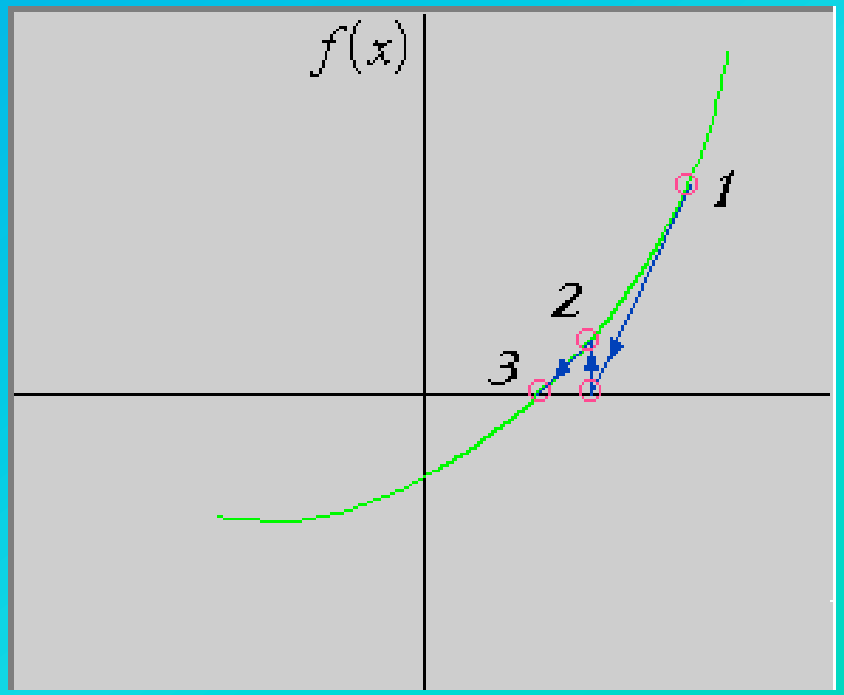
Reminder: Newton-Raphson

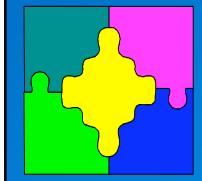
Once the vertex to be moved (m) is chosen all other vertices are fixed and the energy is (locally) minimized by only moving vertex m.

This is done using a single-sided, two-dimensional Newton-Raphson iteration.

Recall: To find the root of a function f(x) in one dimension, Newton-Raphson iterates:

$$x_{t+1} \leftarrow x_t - \frac{f(x_t)}{f'(x_t)}$$





Two-dimensional Newton-Raphson

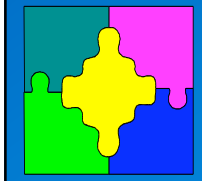
In two dimensions the Newton-Raphson method generalizes as follows:
For a system $f(x^*)$ of equations let $J(f)$ be its Jacobian Matrix.

$$f(x_1,\ldots,x_n)=\begin{bmatrix} f_1(x_1,\ldots,x_n) \\ \vdots \\ f_n(x_1,\ldots,x_n) \end{bmatrix}$$

$$J(f)=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial xn} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

The two-dimensional Newton-Raphson iteration is executed as:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} \leftarrow \begin{bmatrix} x_t \\ y_t \end{bmatrix} - J^{-1}(f(x,y)) \cdot f(x,y)$$



Newton-Raphson Energy Minimization

As we want to minimize the energy $E(x_m, y_m)$ we have to find the solution to

$$\nabla E(x,y) = \begin{bmatrix} \frac{\partial E}{\partial y} & \frac{\partial E}{\partial y} \end{bmatrix}^T = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

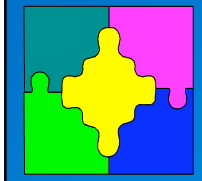
The Jacobian of the energy differential is

$$J(\nabla E) = \begin{bmatrix} \frac{\partial^2 E}{\partial^2 x} & \frac{\partial^2 E}{\partial x \partial y} \\ \frac{\partial^2 E}{\partial y \partial x} & \frac{\partial^2 E}{\partial^2 y} \end{bmatrix}$$

To move vertex *m* to its locally energy-optimal position we therefore iterate

$$p_m^{t+1} = \begin{bmatrix} x_m^{t+1} \\ y_m^{t+1} \end{bmatrix} \leftarrow \begin{bmatrix} x_m^t \\ y_m^t \end{bmatrix} + \begin{bmatrix} \delta x_m \\ \delta y_m \end{bmatrix}$$

$$\begin{bmatrix} \delta x_m \\ \delta y_m \end{bmatrix} = -1 \cdot J^{-1}(\nabla E(x_m^t, y_m^t)) \cdot \nabla E(x_m^t, y_m^t)$$



Evaluation of Force-Directed Layout

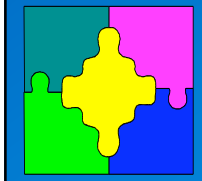
Force-directed Layout is quite useful, because it is a good class of heuristic methods that find nice layouts of general graphs.

These drawings even automatically expose (most of the) symmetries of the given graphs.

However, a number of problems remain:

- These methods are still computationally expensive
- There is no guarantee for a true optimization.
- It is difficult to integrate additional constraints
  (such as preferred node orders, alignments etc.)
- Often edge crossings remain, even in planar graphs
- They treat only idealized graphs,
  node extensions and shapes, label positions etc.
  are not taken into account

- Still, force-directed methods are among the most popular approaches and a large number of variants (with additional forces) have been explored.



Simulated Annealing in Graph Layout

Simulated annealing has successfully applied to the layout of general undirected graphs.

(See R. Davidson and D. Harel. Drawing Graphs Nicely Using Simulated Annealing. ACM Transactions on Graphics, 15(4):301-331, October 1996.)

Neighbourhood function

- Let a single node jump randomly on a circle around its current position
- The maximum jump distance decreases with the temperature

Objective function

$$f(g) = \lambda_1 \cdot \sum_{i \neq j} 1/d(n_i,n_j) \quad \text{Node distribution}$$

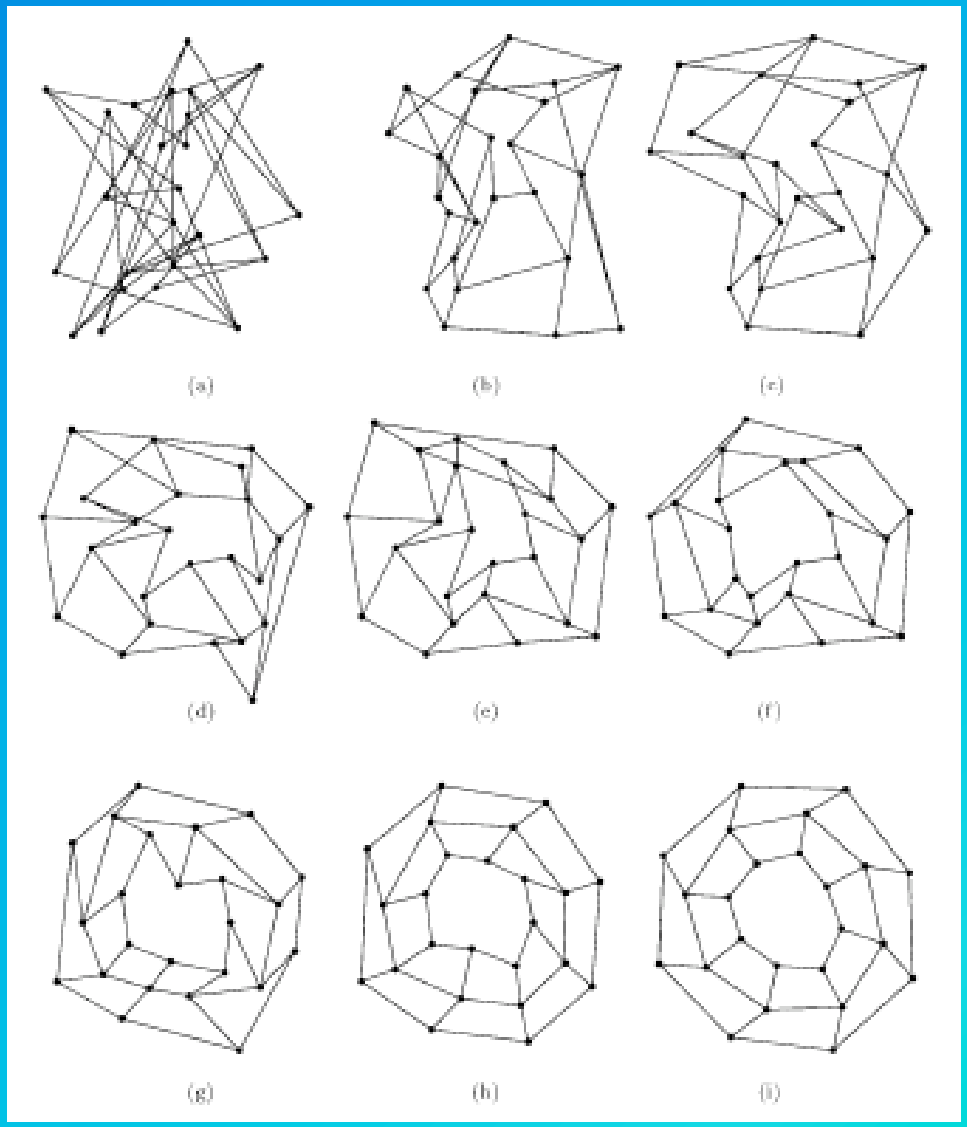
$$+\lambda_2 \cdot \sum_i d_{border}(n_i) \quad \text{Borderline distance}$$

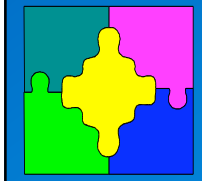
$$+\lambda_3 \cdot \sum_{(n_i,n_j) \in V} d(n_i,n_j) \quad \text{Edge Length}$$

$$+\lambda_4 \cdot num(crossings) \quad \text{Edge Crossings}$$

$$+\lambda_5 \cdot \sum_i \sum_{(n_i,n_j) \in V} d(n_i,e_{i,j}) \quad \text{Edge} - \text{Edge distance}$$

SA Example Animation





Penalty Dependencies

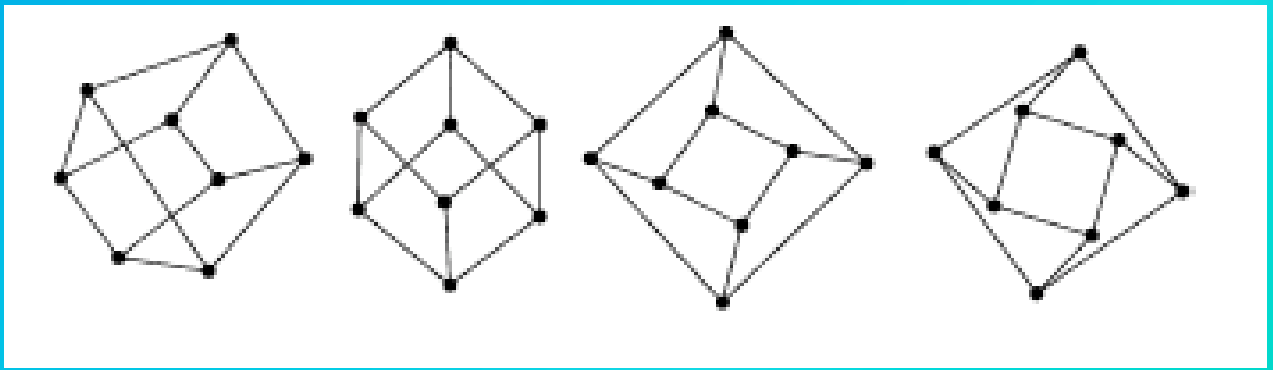
The choice of penalties is crucial for

- quality of final layout.
- efficiency of the method

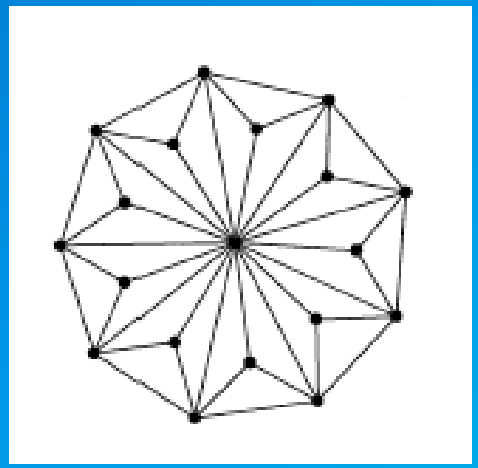
because it determines the ruggedness of the objective function.

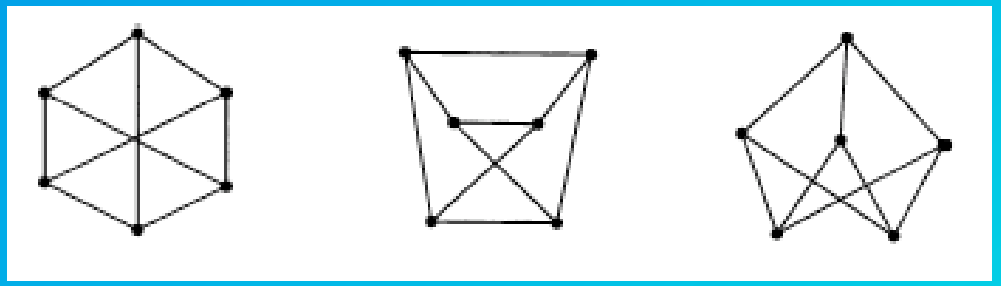
Consider, for example, the edge crossing component which is not a smooth (or continuous) function. For smooth objective functions, SA is relatively more efficient.

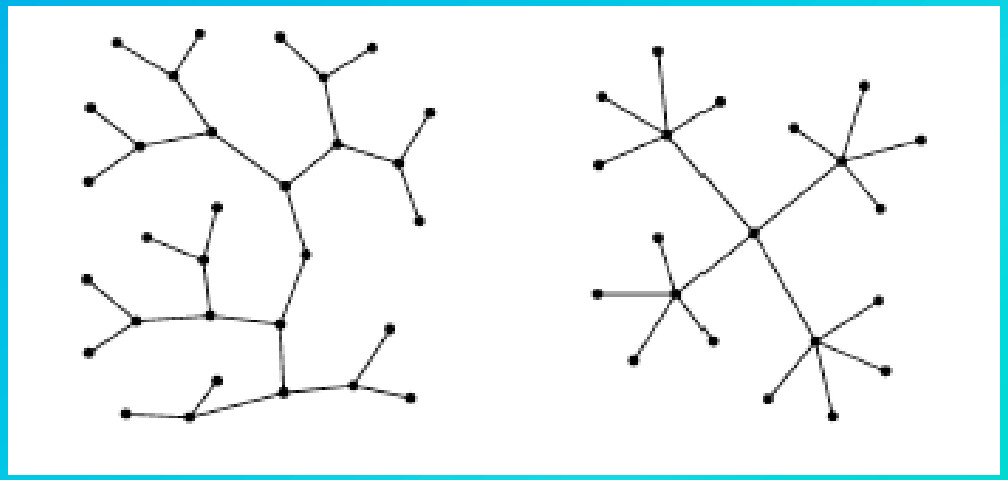
Variations Depending on Penalties

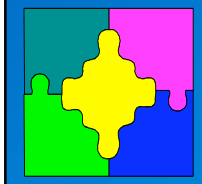


More SA-GD Examples

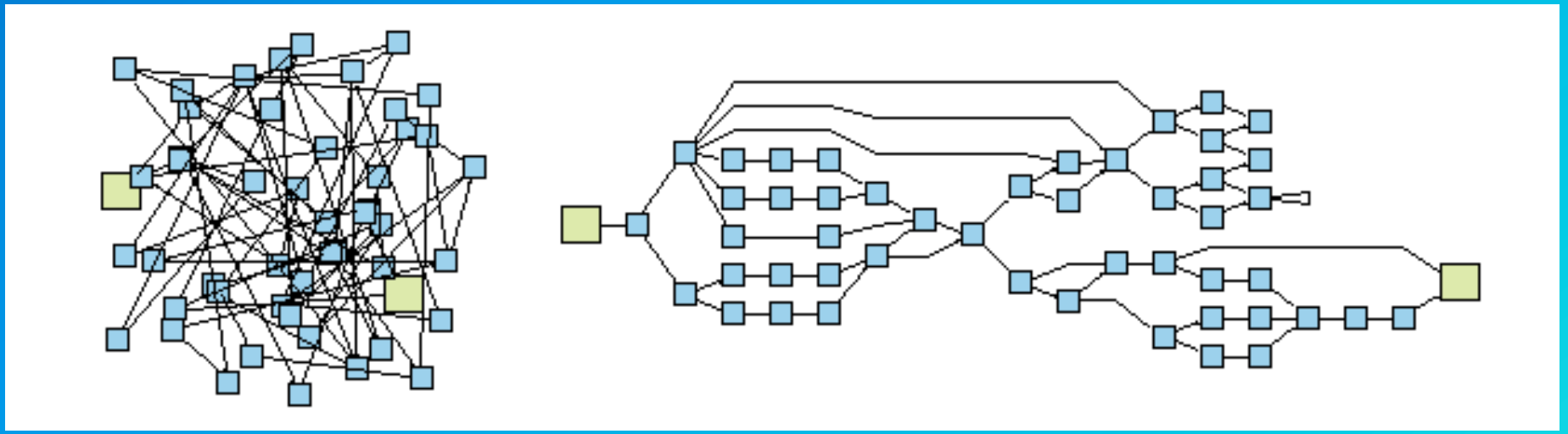








DAG Layout

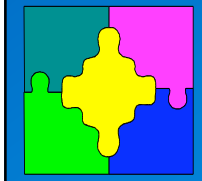


Directed acyclic graphs can be drawn in layers.

DAG layout is much simpler and can be solved in three steps:

(1) assign each node to a layer (layering)
(2) insert new “virtual nodes” so that no edge crosses a layer
(3) sort the nodes in each layer (ranking)
(4) determine the absolute node coordinates from the rank information

Step 1+4 can be solved with integer programming techniques.
Real implementations use network simplex.



Layer Assignment

can be formalized as an integer programming problem.

The objective is to

- minimize the (weighted) length of edges
- keeping a (predefined) minimum distance between nodes.

Let $G = (V, E)$ be a DAG.

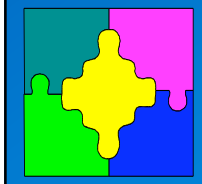
Let $\delta(v, w)$ be the required minimum distance between nodes v, w

Let $\omega(v, w)$ be the edge weight on the edge (v, w)

Let λ(v) be the layer number assigned to a node v

minimize $\sum_{(v,w) \in V} \omega(v, w) \cdot (\lambda(w) - \lambda(v))$

subject to $\lambda(w) - \lambda(v) \geq \delta(v, w)$



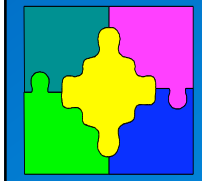
Ranking within Layers

- cannot be readily formalized as integer programming
- must be solved with heuristic methods
- Note that the total number of crossings can be minimized by minimizing the number of crossings between adjacent layers independently

```
procedure rank
        order=initial_random_order;
        best=order;
        for i=0 to max_iterations do
                median_order(order,i);
                transpose(order);
                if crossing(order)<crossing(best) then
                        best=order;
        end
        return best;
end.
```

median order sweeps layers in increasing order for i even
in decreasing order for i odd. The rank of a node in the current layer is computed as the median of all adjacent nodes' ranks in the following layer .

transpose tries to minimize the number of crossings between the current layer and the following layer by swapping neighbouring nodes.



Computing Node Coordinates

can be formalized as an integer programming problem.

The objective is to

- minimize the (weighted) length of edges
- keeping a (predefined) minimum distance between nodes.

Let $\rho(v,w)$ be the required minimum separation between nodes v,w

Let $\omega(v,w)$ be the edge weight on the edge (v,w)

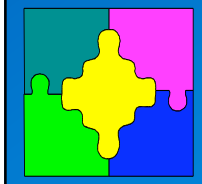
= relative importance of straightening this edge

Let y(v) be the absolute y-coordinate assigned to a node v

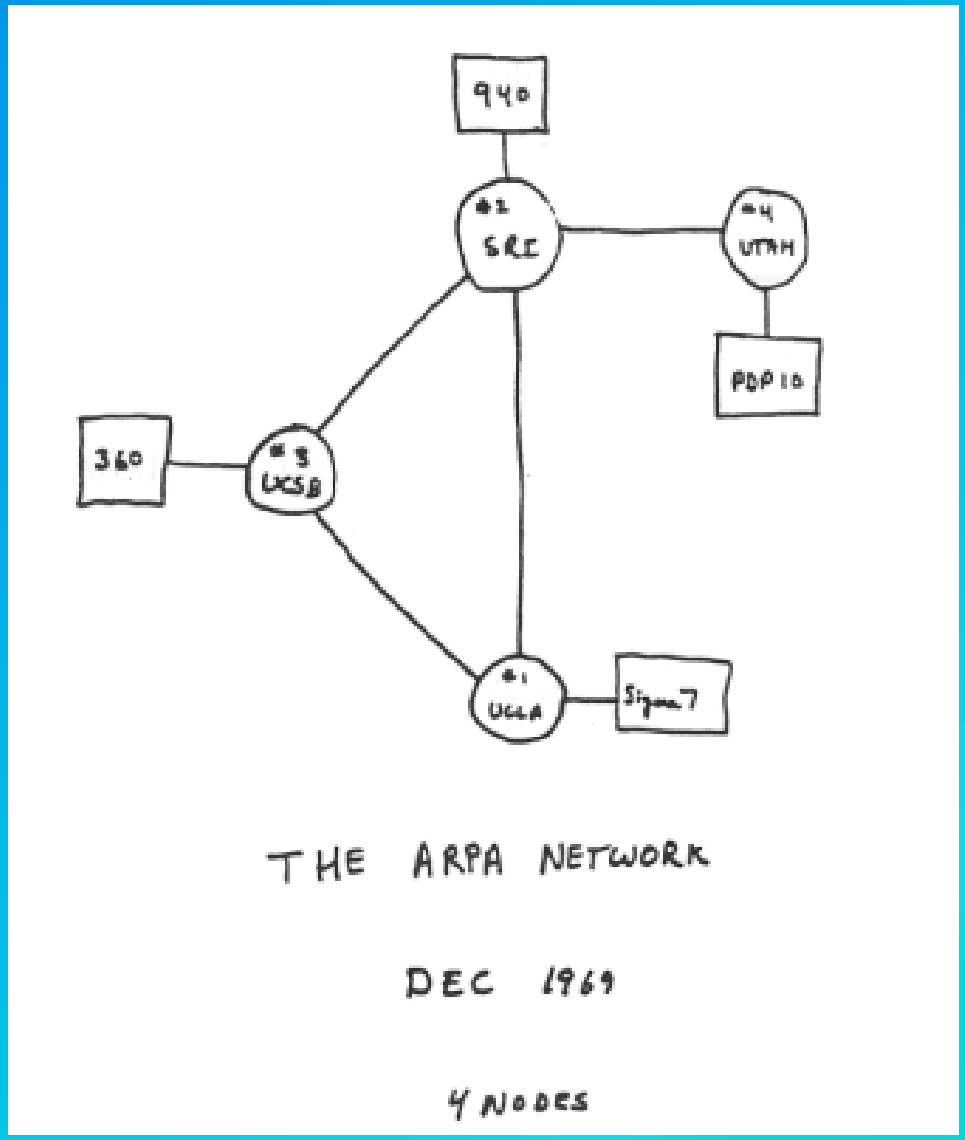
minimize $\sum_{(v,w)\in V} \omega(v,w) \cdot | \, y(w) - y(v) \, |$

subject to $\forall_{\text{node a is direclty above b in some rank}} \; y(b) - y(a) \geq \frac{width(a) + width(b)}{2} + \rho(a,b)$

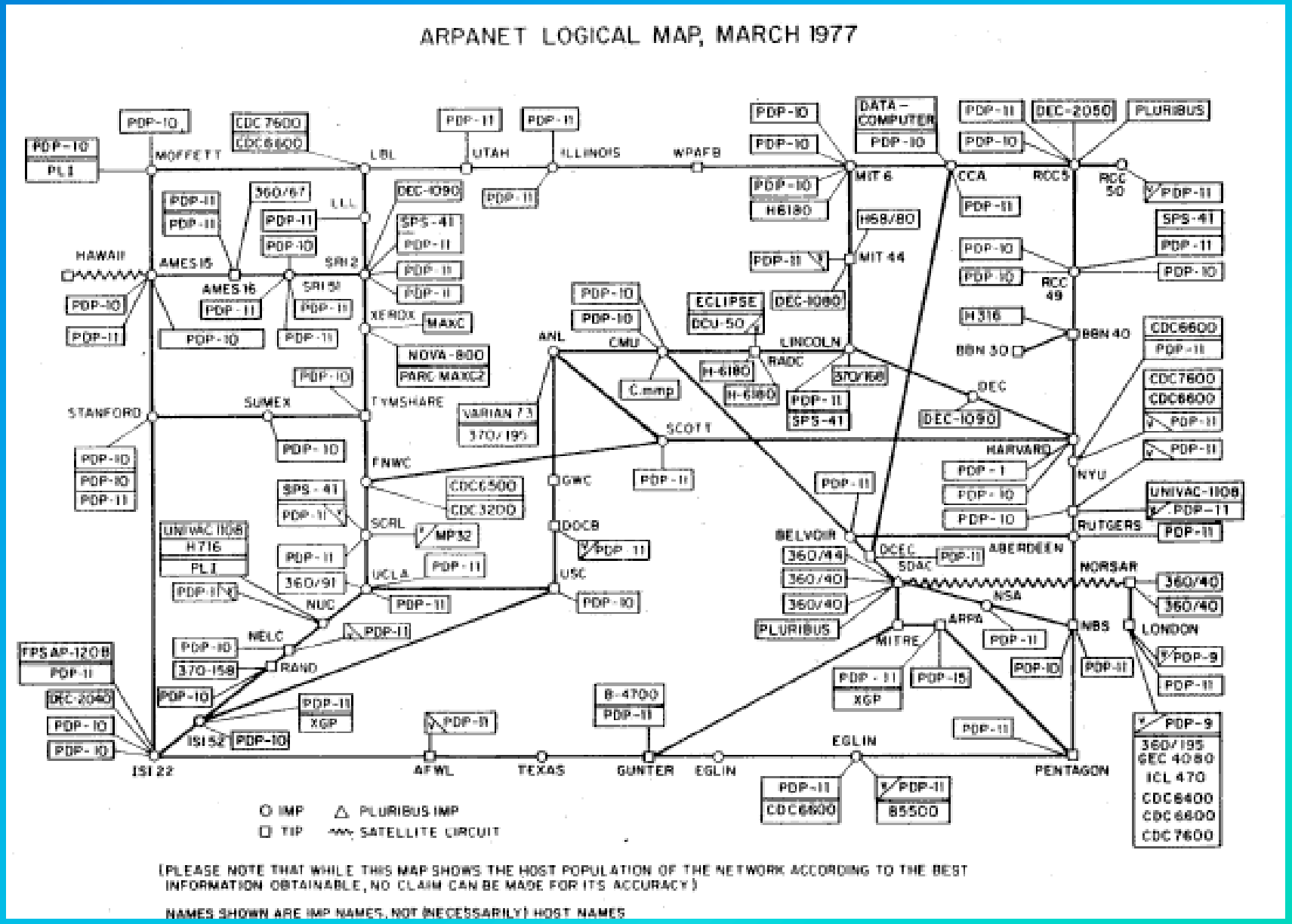
Note: the absolute value can easily be transformed using standard 0-1 IP techniques.

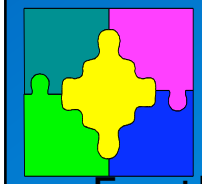


Drawing Small Graphs...



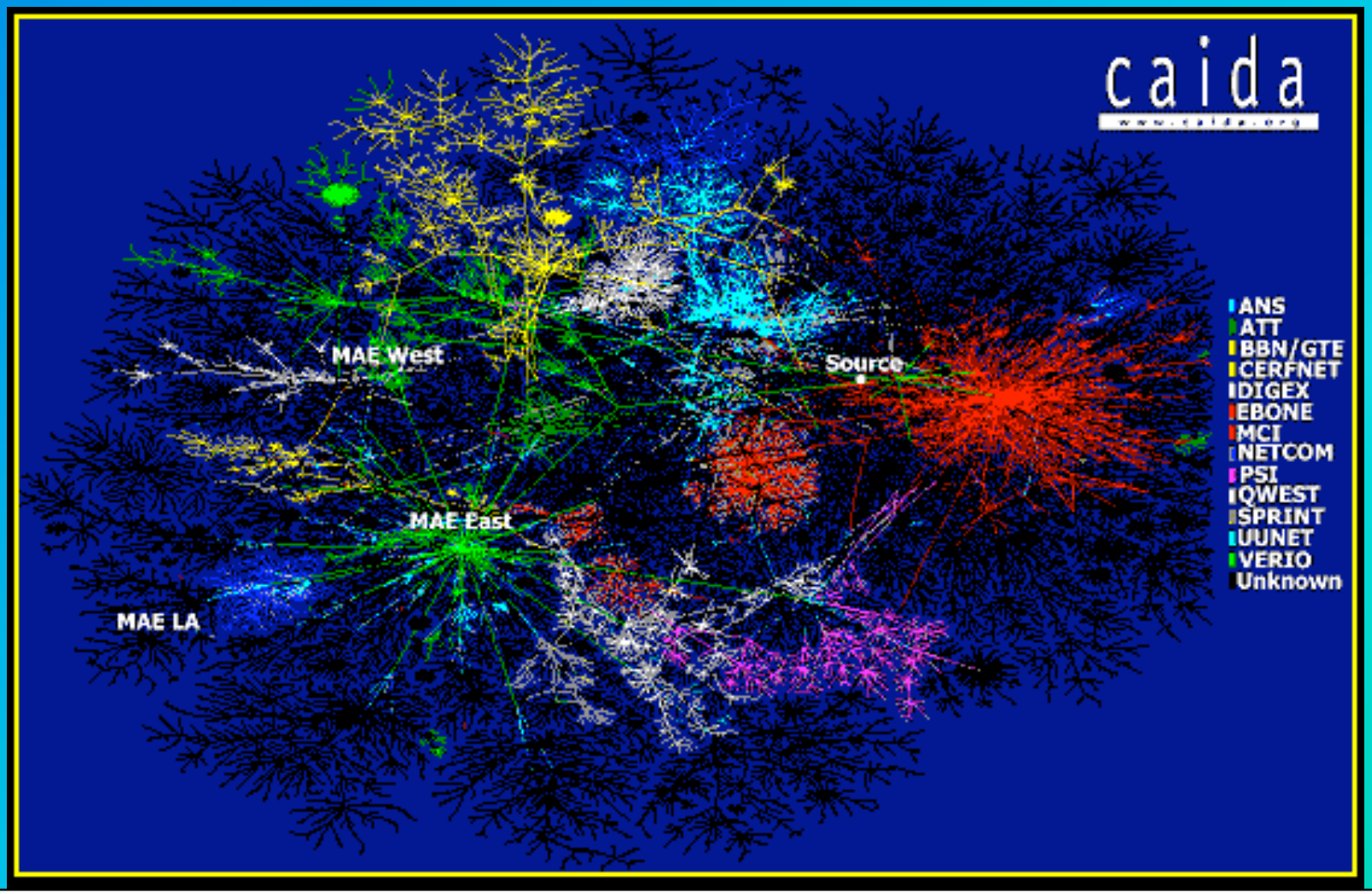
Drawing Larger Graphs

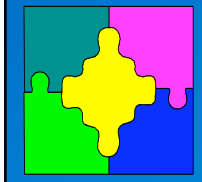




... Drawing of Huge Graphs

For the layout of very large graphs, such as visualization of software repositories or web structure, the run time of the layout algorithm is extremely critical and the methods we have shown cannot be used. In these cases “beauty” is only secondary. The main interest is to get an overall view of the relational structure, in particular identifying clusters of nodes.





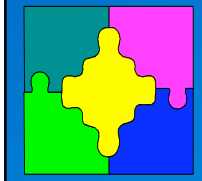
Fast Layout Methods for Large Graphs

For the drawing of very large graphs much faster clustering algorithms must be used and a finer and more detailed layout may be generated after obtaining a smaller graph by zooming in (and/or removing irrelevant nodes).

For examples of such algorithms see:

Self-organizing Graphs.
B. Meyer.
In International Symposium on Graph Drawing 1998. Springer LNCS 1547

Self-organizing Maps for Drawing Large Graphs.
E. Bonabeau and F. Henaux.
Santa Fee Institute Technical Report 98-07-066.
Reprinted in Information Processing Letters.



Challenges in Graph Layout

1. Drawing very large graphs
2. Taking additional constraints into account
3. Drawing more complex diagrams (e.g. State Charts)
4. Interactive exploration of diagrams