Smart Graphics: Methoden 4
point feature labeling, graph drawing

Vorlesung „Smart Graphics”

Andreas Butz, Otmar Hilliges

Dienstag, 6. Dezember 2005

Point-feature label placement

- automated layout planning
- point-feature label placement (PFLP)
- exhaustive techniques
- heuristic techniques
  - control strategies
  - pruning heuristics
  - discrete gradient descent
- Hirsch’s algorithm
- stochastic approaches
  - simulated annealing

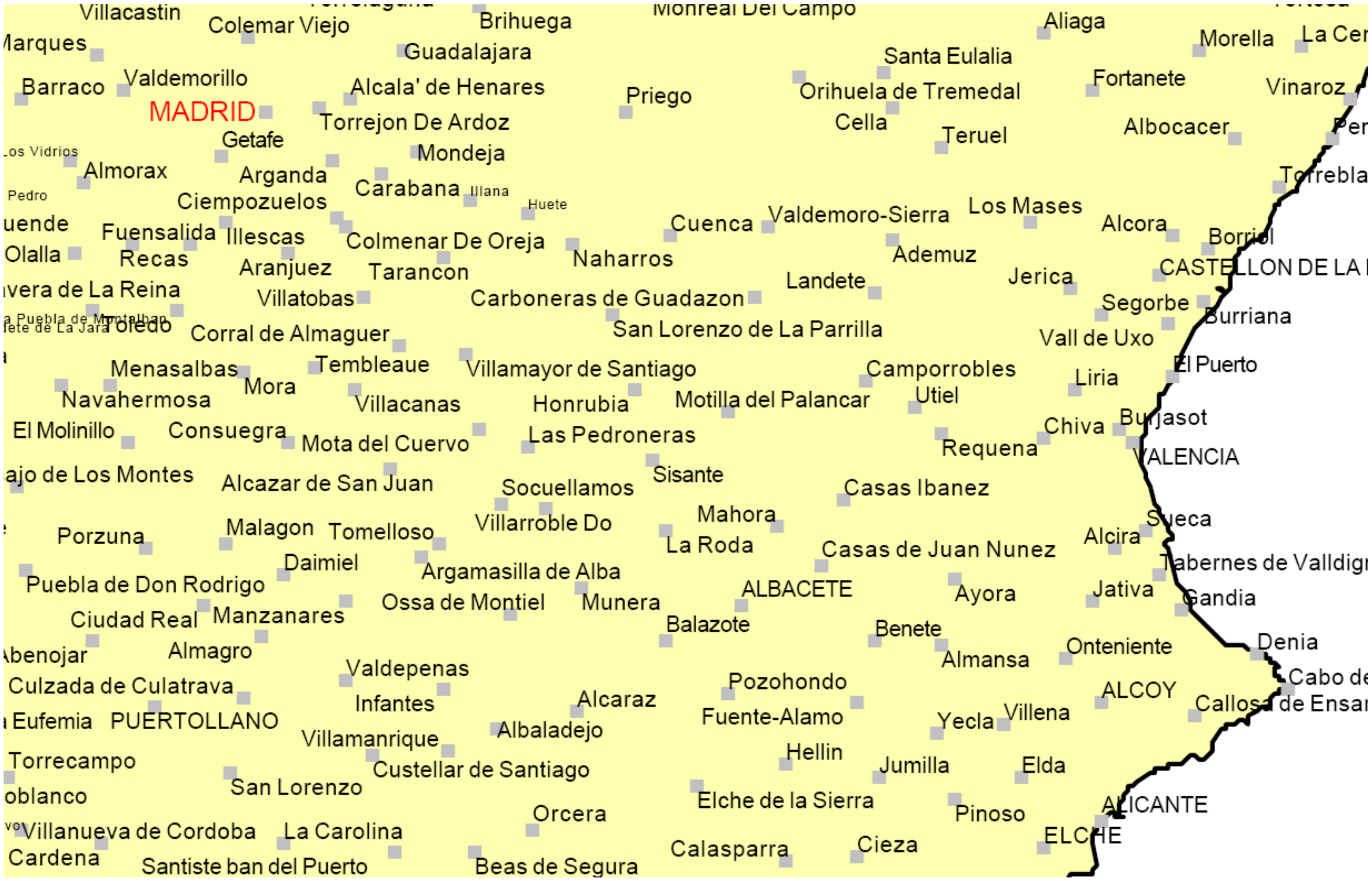
Automated layout planning

- **aspects of a graphical interfaces:**
  - content: information to be presented
  - layout: visuospatial properties of the presentation
  - interaction: how the user and presentation interact
- **layout planning and the need for automation:**
  - hard to do even for experienced graphics designers
  - production times are getting shorter
  - content to be presented may be changing with time
  - may need to make a tailor a presentation to the user
  - may have resource limitations:
    - interface constraints (e.g. PDA resolution)
    - computational resources (e.g. communications bandwidth)

Classes of layout problems

- **text formatting**
  - e.g. enumerations, paragraphs, footnotes, columns, captions
- **text layout**
  - e.g. column width, caption placement
- **2D layout**
  - label placement (place text in a static graphic)
  - graph drawing (layout a structure with fixed topology)
  - text and graphics layout (configuring a chart or multimedia document)
- **3D layout**
  - 3D versions of the the 2D problems (especially visualization)
  - additional issues of lighting and camera configuration

Cartographic label placement

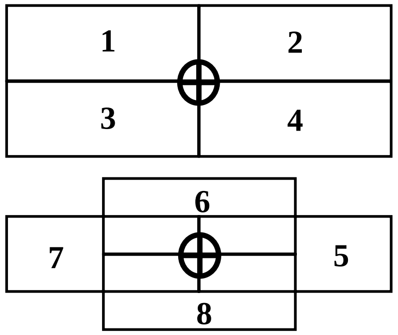


http://www.evermap.com/evername/SpainMap.pdf

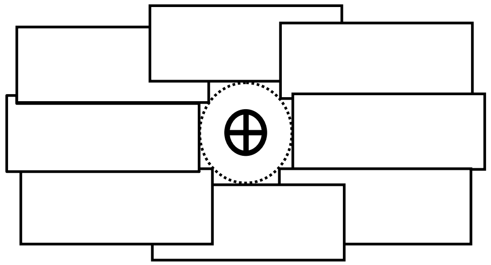
Cartography: problem characterization

- **tagging graphical objects with text labels is a fundamental problem in many graphical user interfaces (and cartography)**
  - 2D graphics: charts, plots, visualization interfaces
  - 3D graphics: visualization and augmented reality
- **in cartography there are three label-placement tasks;**
  - area-features, line-features, point-features
- **human performance is not good:**
  - 20-30 label placements per hour
  - font, style, size, and placement takes 50% of high quality map production
- **complexity issues:**
  - NP-hard, therefore existing algorithms are either exponential or incomplete
  - complexity arises due to the potentially global consequences of label-label overlap

PFLP: search space



- discrete:
  - cartographic standards
  - 4-8 possible label assignments
  - number indicates preference of label assignment
- continuous:
  - specify a circle surrounding the point-feature
  - label must touch the circle without intersecting it
- point selection:
  - in some contexts the only solution to an overlap is not to include a label
  - referred to as the **point selection problem**
  - more typical to prefer “overplots” (and thereby guarantee completeness)



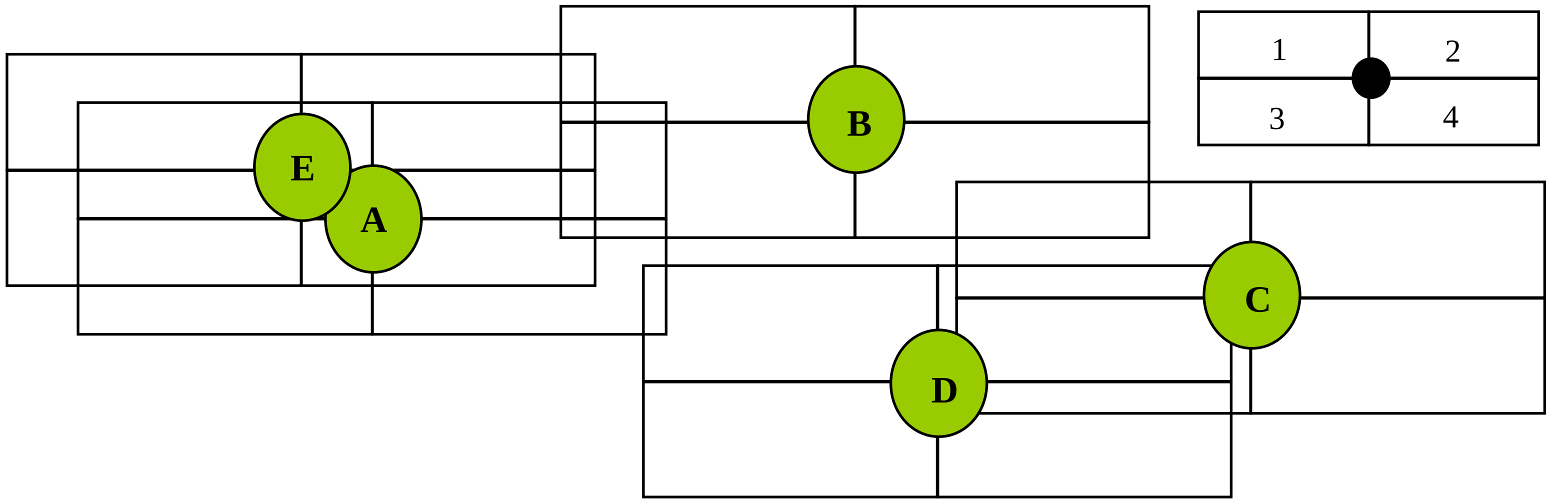
PFLP: objective function

- **state: potential labeling of points**
- **objective function:**
  - assign a value to a possible state based on the relative quality of the labeling
  - aspects of point-feature labeling objective function:
    - **overlap**: the number, or area, of overlap with other text labels or graphical features
    - **preferences**: preferences within a canonical set of labelings
    - **point-features not labeled**: the number not labeled for point-feature selection
- **discrete PFLP is a combinatorial optimization problem:**
  - exhaustive, heuristic (and continuous heuristic), and stochastic solution techniques
  - consider the algorithms and the empirically established efficiency

Exhaustive search techniques

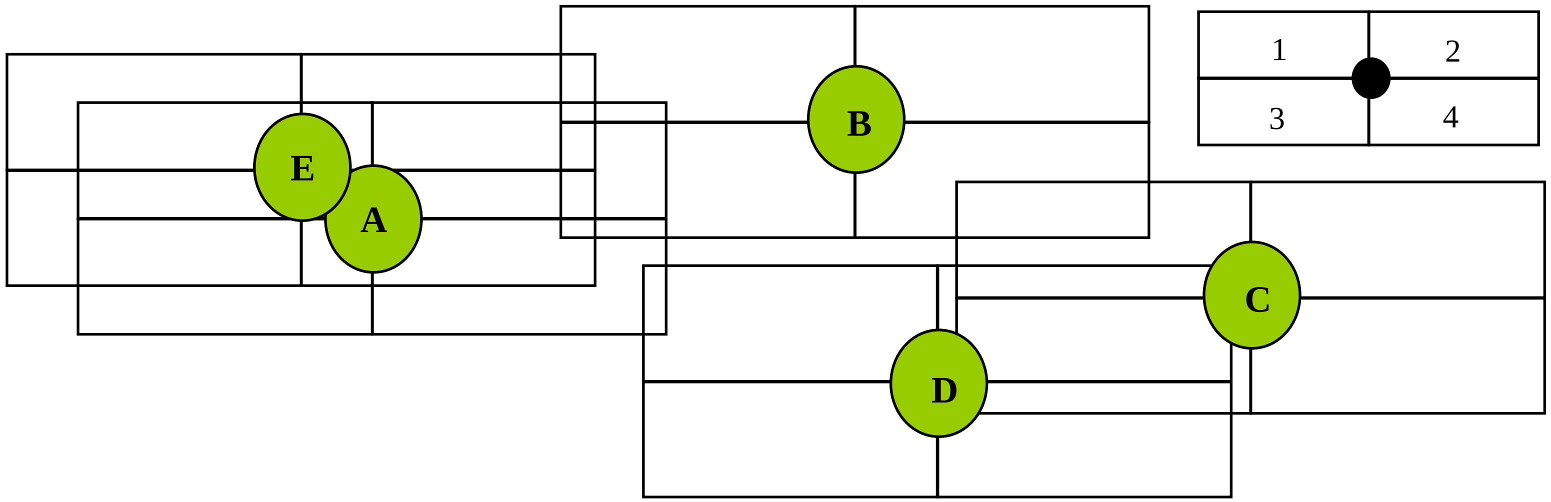
- exhaustive constraint satisfaction can be classified according to the way in which they:
  - perform backtracking
  - "prune" the search space
- examples (as relevant to PFLP):
  - **variable ordering**: minimize backtracking by making the most constrained choices first; by labeling the most difficult point-features first we are likely to reduce backtracking in the future
  - **value ordering**: when labeling a point-feature, pick the value that is least likely to give rise to a conflict; can assess this based on the point-features in the immediate neighborhood
  - **source of failure**: when an exhaustive search procedure backtracks, it usually does so to the most recently placed label, better to identify the source of the failure and rectify that
  - **pruning heuristics**: eliminate label positions which exceed a predefined density threshold (of conflict with other label positions); relax the density threshold if a solution at the current level cannot be found

A simple PFLP example



- total size of search space $b^d = 4^5 = 1024$
  - b = number of possible labeling for a point feature
  - d = number of point features to be labeled
- if the search proceeds A, B, C, D, E with preference 1, 2, 3, 4 then since A1 conflicts with all E labelings, we know we'll explore at least 4^4 labelings (256)
- we can examine how the heuristics and pruning techniques might affect this

A simple PFLP example



variable ordering

- calculate how many of a point-feature's positions yield a clash, and (number of other positions they clashes with):

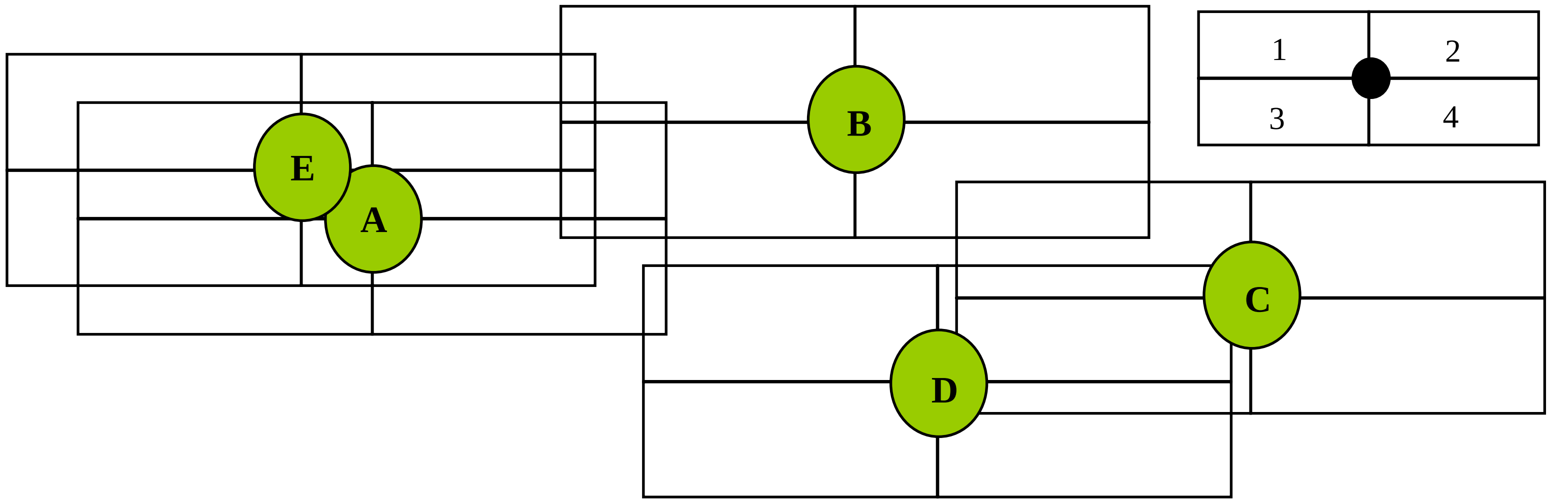
  A = 4 (7)
  B = 3 (5)
  C = 2 (3)
  D = 3 (3)
  E = 4 (6)
- assign labels in order: A, E, B, D, C

value ordering

- Pick the least conflicting label first:

  A = A3(2), A4(3), [A1(4), A2(4)]
  B = B2(0), B4(1), B1(2), B3(4)
  C = [C2(0), C4(0)], [C1(2), C3(2)]
  D = D3(0), [D1(1), D4(1)], D2(2)
  E = E1(1), E3(2), E2(4), E4(5)
- <A3, B2, C2, D3, E1> is a valid and non-overlapping labeling

A simple PFLP example



source of failure

- on a depth-first search for a labeling, successive labelings of E fail for the set of labelings
  <A1, B1, C1, D1, …>
- conventional depth-first backtracks to relabel D, but since all the labelings of E conflicted with A1, this is the source of failure which forces re-labeling of A.

density threshold

- find value of “density” threshold, pick the maximum of these (which is 2)

  A = A3; D = D1, D2, D3, D4;

  B = B1, B2, B4; E = E1, E3

  C = C1, C2, C3, C4;
- Search space reduced to 1´3´4´4´2=96 but still contains valid labelings.

Greedy algorithms

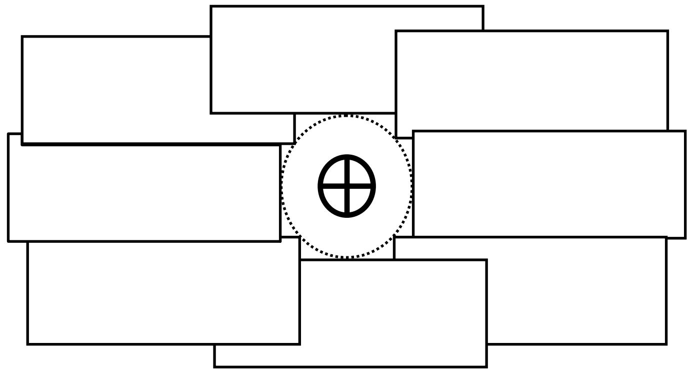
- no backtracking, just do the best label assignment next
- any point that cannot be labeled without a conflict:
  - leave the point out (point selection)
  - assign overlapping label, and allow the resulting obscuration
  - appeal for user assistance (human oracle)
- typically gives rise to non-optimal labeling, at a low cost
- greedy algorithm performance can be improved by adding/modifying the objective function, for example, the value ordering method used to improve exhaustive search.

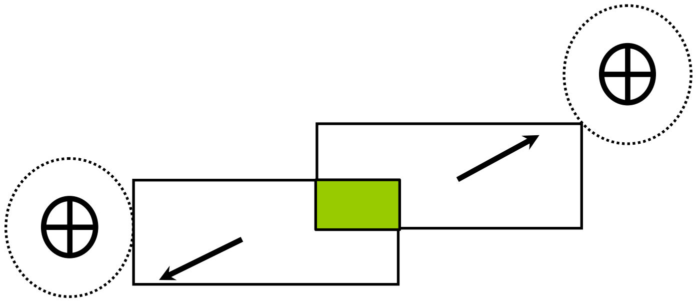
Discrete gradient descent

- Another method related to greedy algorithms is to consider the change in the objective function resulting from a change in the labeling of any single point-feature.
  – For each feature, randomly place its label.
  – Repeat until no improvement in gradient function:
  – For all single changes in label position: calculate the corresponding change in the objective function.
  – Perform the label repositioning that yields the greatest improvement in the objective function.
- The problem (as with all gradient descent approaches) is that it is only a local search method and will get stuck in local maximum.

Hirsch’s algorithm

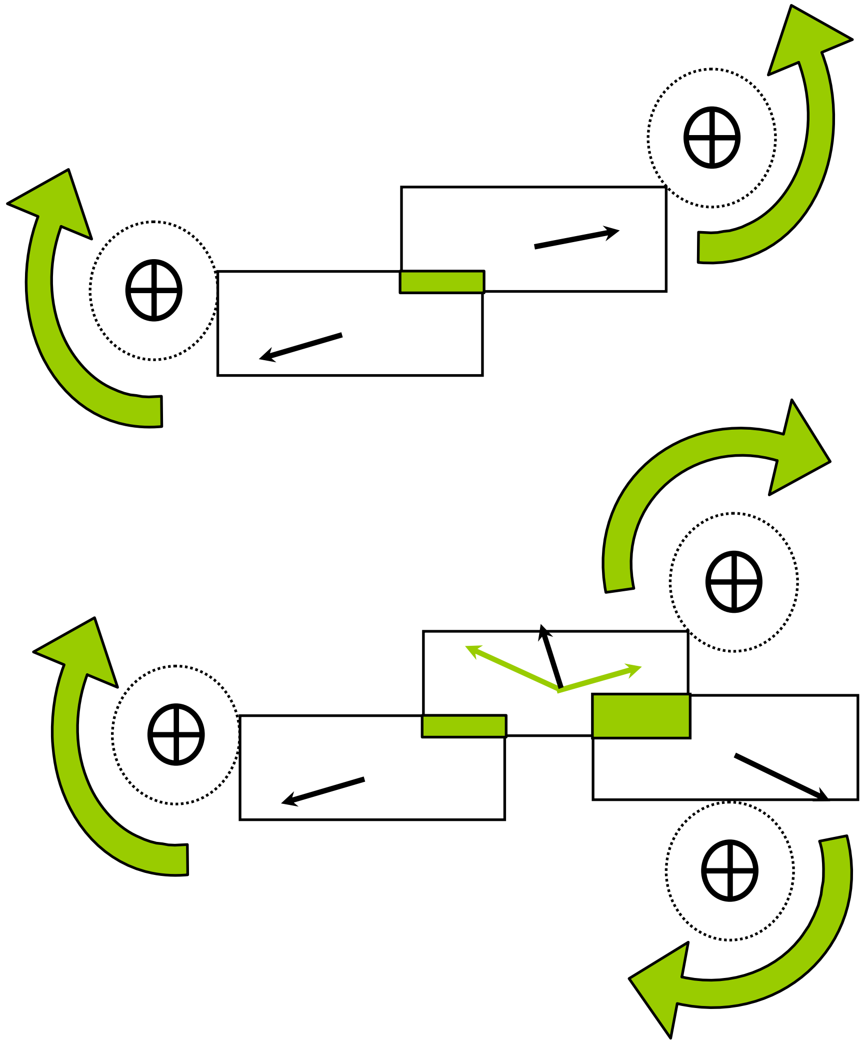
- continuous placement model, bounding rectangle of a label touches a circular region surrounding the point feature
- when the edge of the label is tangential to the circular region the label may slide
- initially each label is placed to the right of its feature
- the direction to move the labels to resolve the overlap is derived from the overlap region
- [S. A. Hirsch: *An Algorithm for Automatic Name Placement around Point Data*, The American Cartographer, 9(1), pp. 5–17,1982.]





Hirsch's algorithm (cont)

- each overlap between two regions is converted into a pair of vectors which point into its respective label
- if a label has multiple overlaps then the vectors are summed
- two types of motion:
  - around the circle an amount proportional to the resultant vector
  - discontinuous jump to a point indicated by the resultant vector
- problems:
  - local minima
  - vector summation errors



Stochastic search: simulated annealing

- the heuristic search methods that we've considered are plagued the problem of local minima
- simulated annealing is a stochastic search technique that addresses this by incorporating a probabilistic element into the search
- outline algorithm:
  - randomly assign values to variables, initialize temperature T
  - repeat until rate of improvement falls below a threshold:
  - assign a randomly selected variable to a different one of its possible values (randomly select which one)
  - compute ΔE the change in the objective function that results
  - if the change is for the worse, then undo with a probability $P = 1.0 - e^{-\Delta E/T}$
  - decrease T according to the annealing schedule

Simulated annealing and PFLP

- **initial configuration**: possible to start with randomly placed labels, or use SA to post-process the results of a local method.
- **objective function**: ΔE should be quick to evaluate as computed many times (e.g. the number of overlapping labels).
- **configuration changes**: how to choose which label to reposition (for example, choose randomly from all labels, choose randomly from those experiencing a conflict)
- **annealing schedule**: empirical issue, Christensen initializes T so that P = 2/3 when ΔE=1. Maximum of 20n labels changed before T reduced by 10% (n is the number of point-features)

Introduction to graph drawing

- characterize graphs and motivations for graph drawing
- review relevance of approaches to PFLP to graph drawing
- characterize the cognitive issues involved in the design and application of effective graph drawing techniques
- review the main classes of graph drawing algorithms
- consider in detail a few graph drawing algorithms
  - binary tree drawing algorithms
  - visibility representations as an intermediary to graph drawings
  - local and global search approaches to graph drawing

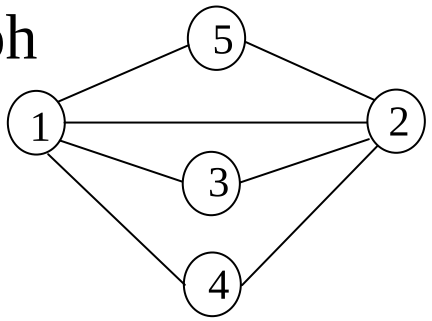
Definitions: what is a graph?

- a graph ***G = (V, E)*** consists of a finite set ***V*** of vertices and a finite multiset ***E*** of edges – unordered pairs ***(u,v)*** of vertices (or *nodes*)
- the ***end-vertices*** of an edge ***e = (u,v)*** are ***u*** and ***v***, and we say that ***u*** and ***v*** are adjacent to each other and that ***e*** is incident to ***u*** and ***v***
- a ***directed graph*** (a ***digraph***) is defined similarly except that the elements of ***E***, called ***directed edges***, are ordered pairs of vertices
- a ***directed edge***, ***(u,v)***, is an ***outgoing edge*** of *u* and an ***incoming edge*** of ***v***
- graphs with edges between all vertices are ***fully connected*** and graphs with few edges between nodes are referred to as ***sparse***
- a ***drawing*** of a graph is ***planar*** if no two distinct edges intersect
- a planar drawing partitions space into topologically connected regions called ***faces***, and the unbounded face is referred to as the ***external face***

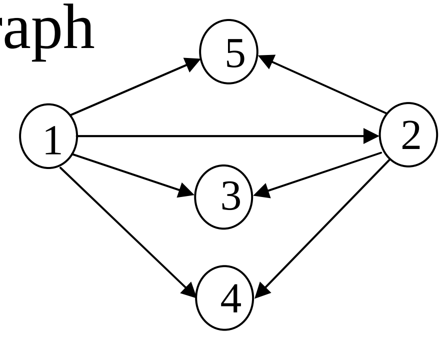
What is a graph (the pictorial guide)?

```
G = (V,E)
V = {1, 2, 3, 4, 5}
E = {(1,2), (1,3), (1,4),
     (1,5), (2,3), (2,4),
     (2,5), (3,4)}
```

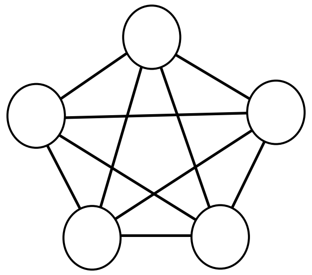
graph



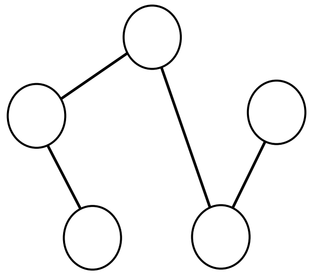
digraph



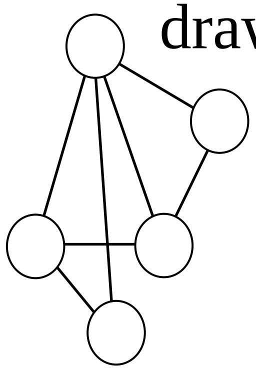
fully connected



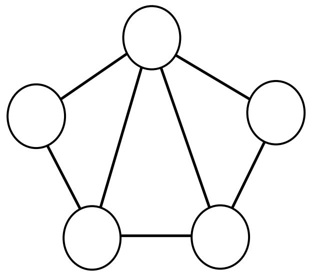
sparse



non-planar drawing

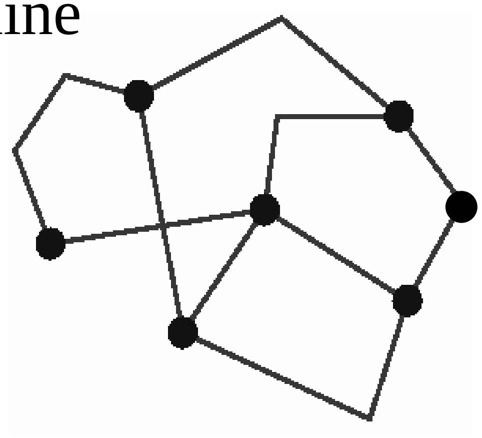


planar drawing

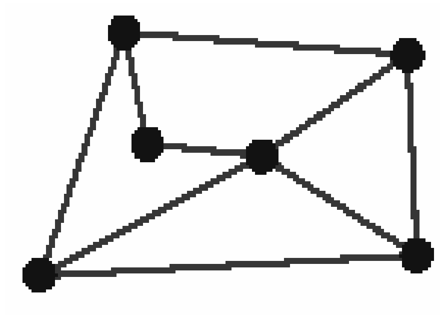


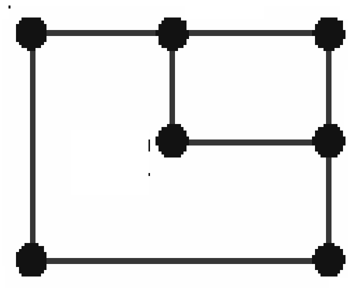
What is a graph drawing?

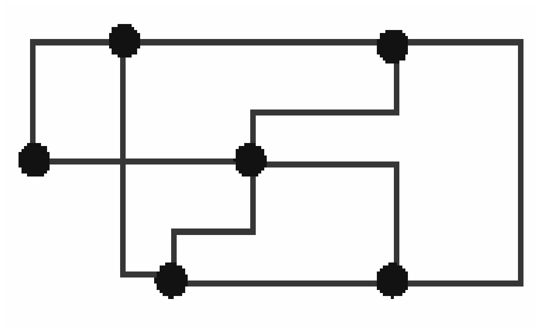
polyline



planar straight-
line







planar rectilinear

orthogonal

Motivations: application and problems

- situations in which actual (and derived) data is naturally represented as a graph are numerous:
  - WWW: global networks, sitemaps, history diagrams
  - software engineering: data flow diagrams, OO class hierarchies
  - artificial intelligence: semantic networks, terminological networks
  - management: organisational charts, PERT charts
- actual problem: **graph visualization or graph drawing**
  - the local and global properties of the relational data to be displayed (e.g. visual similarity of isomorphic subgraphs)
  - the interaction requirements (e.g. edits) and the graph drawing algorithm, redrawing within time constraints of interaction

Requirements: visual aesthetics

- global:
  - **area**: minimize the total area
  - **aspect ratio**: minimize the aspect ratio (longest:shortest ratio of bounding rectangle)
  - **symmetry**: display the symmetries of the graph in the drawing
- edges:
  - **total edge length**: minimize the sum of the edge lengths
  - **maximum edge length**: minimize the longest edge length
  - **uniform edge length**: minimize the variance in the edge lengths
  - **crossings**: minimise the number of crossings
  - **angular resolution**: maximise the smallest angle for two edges incident on the
- bends:
  - **total bends**: minimize the total number of bends
  - **maximum bends**: minimise the maximum number of bends on an edge
  - **uniform bends**: minimise the variance in the number of bends on edges

Requirements: constraints

- placement constraints
  - place a vertex close to the centre of a drawing
  - place vertex on the external boundary of a drawing
  - cluster a group of vertices together in a spatial region of a drawing
  - all the above, but with subgraphs rather than vertices
- spatial sequencing
  - draw a particular path in a particular direction
- shape
  - draw a graph within a specified shape
- efficiency
  - draw a graph within a specified time bound
  - **anytime** behaviour is often desirable

drawing convention

+

visual aesthetics

+

constraints

$\Rightarrow$

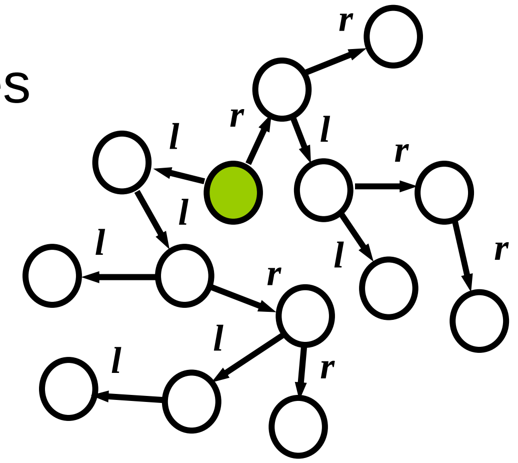
graph drawing (visualization) problem

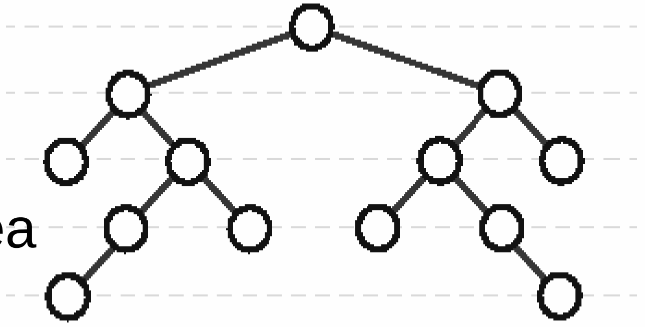
Relevance of PFLP techniques

- Our approaches cast PFLP problem as an optimisation problem
  - mostly, as a combinatorial optimisation problem (use local and global search)
  - Hirsch’s algorithm is a counter example
- PFLP local search techniques may be applied but it’s less natural to cast graph drawing as a discrete search problem
- stochastic PFLP was a declarative approach
  - establish the requirements of a well labelled set of point features
  - state the property of the solution
  - use (relatively) general purpose techniques to find a solution
- graph drawing can be declarative, but mostly algorithmic
  - what do we mean by this declarative v algorithmic contrast…..

Algorithmic approaches: trees

- **rooted trees**:
  - used in data structures and simple hierarchies
  - directed acyclic graphs
  - all edges directed away from the root
- **binary rooted tree**:
  - vertices have 1 in-coming edge (except root)
  - vertices have zero, one, or two out-going edges
- **general drawing requirements**:
  - planar and straight-line, minimizing the area
  - upward (parents above children)
  - display symmetries and isomorphic subtrees
  - nodes in a generation are horizontally aligned





Algorithm 1: layering

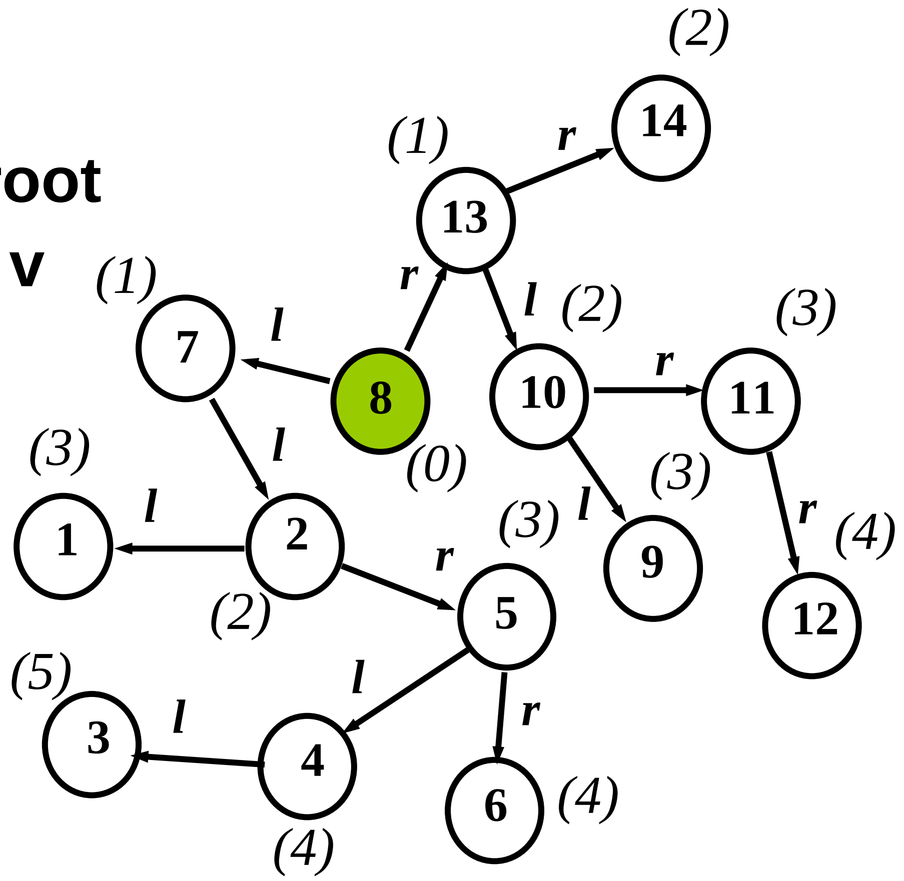
"**layered drawings** place vertices according to their *depth* from a reference node, typically this prescribes the y-coordinates of the vertices"

For example:

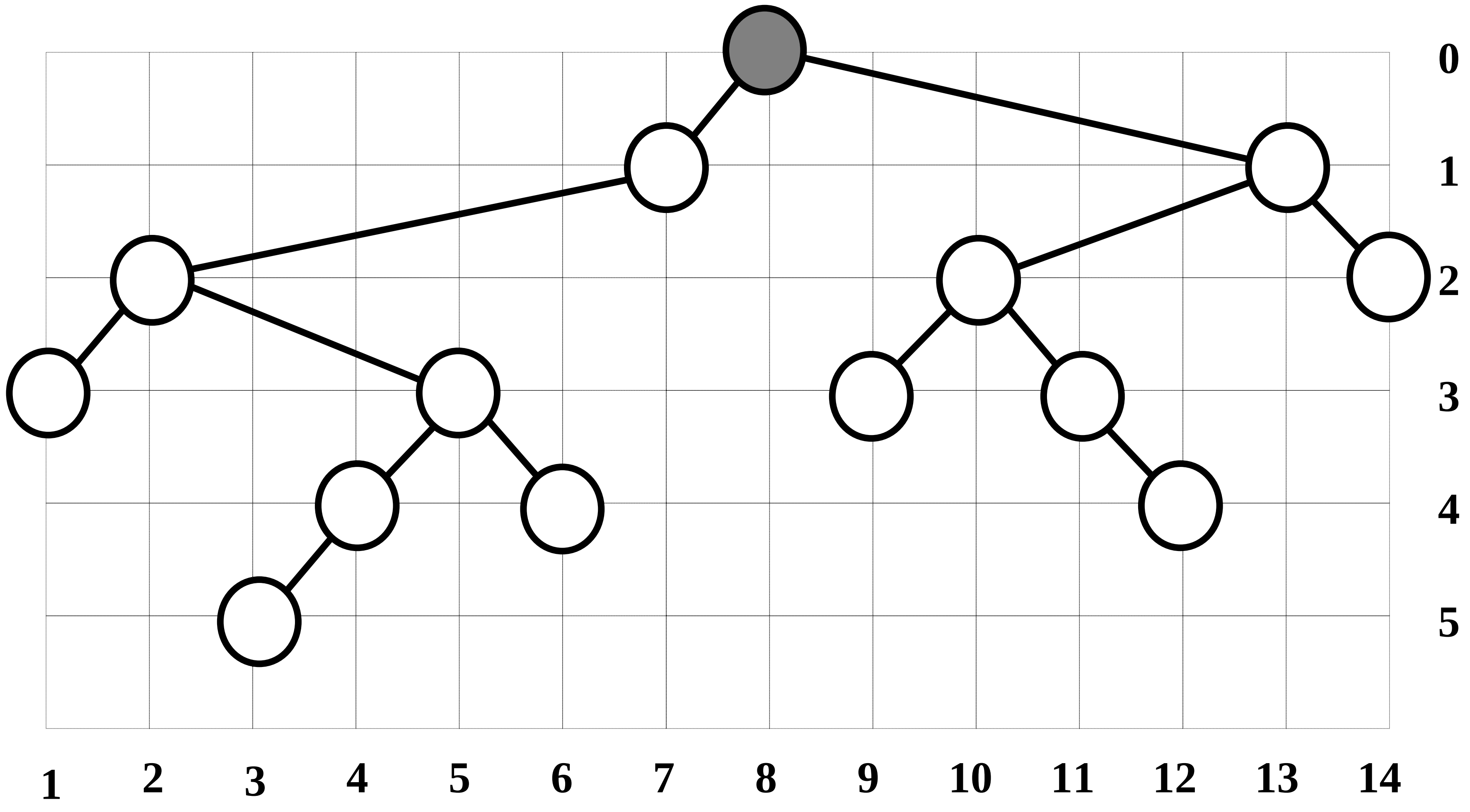
– set**: y(v) = distance from root**

– set**: x(v) = inorder rank of v**

inorder traversal*: defined for binary trees as the recursive traversal of the left subtree of the root, followed by the root, and then the right subtree.*

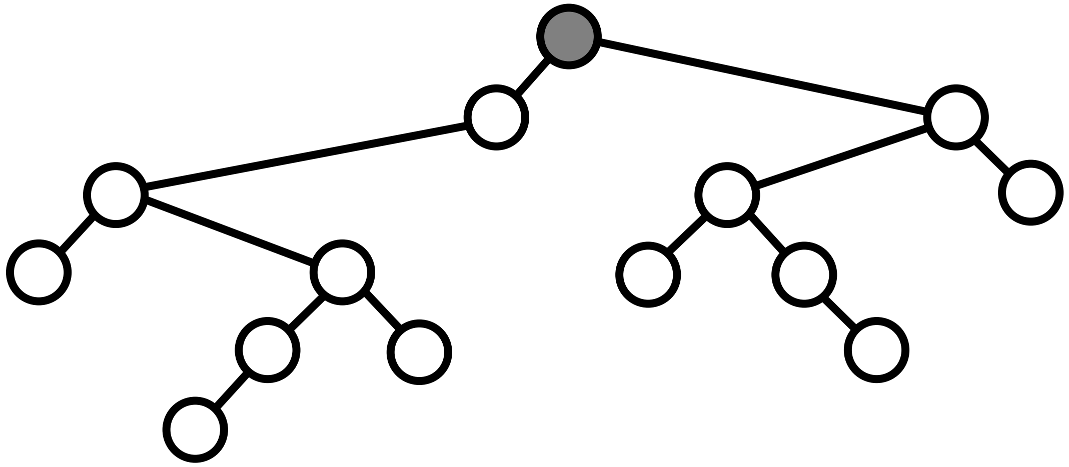


Algorithm1: results



Algorithm 1: properties

- level grid drawing
- display of symmetries and of isomorphic subtrees
- parent always between left and right children
- parents are not always centred on children
- width <= n - 1



Algorithm 2: divide and conquer

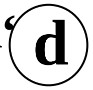
"Divide and conquer approach: (1) split the tree into subtrees; (2) recursively draw the subtrees; (3) glue the resulting drawings together"

For example:

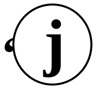
- draw the left subtree, then draw the right subtree
- place the drawings of the subtrees at a distance of 2 between their horizontal extents
- place the root one level above and half-way between children
- if there is only one child, place the root at horizontal distance 1 from the child

Algorithm 2: results

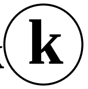
1. **At "a" draw "b-subtree":**
    1. **At "b" draw "d“**

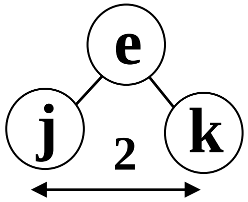
    2. **At "b" draw "e-subtree":**
        1. **At "e" draw "j“**

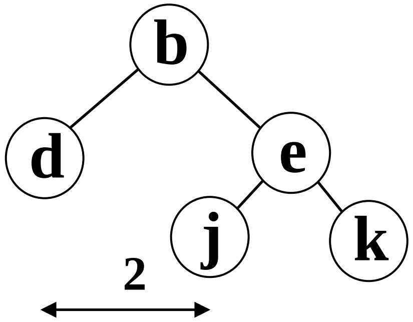
        2. **At "e" draw "k**

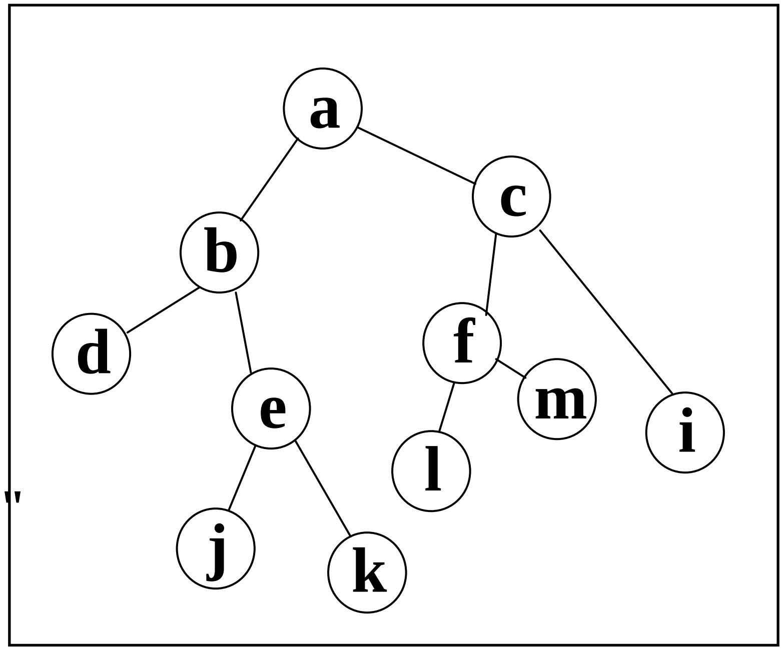
           

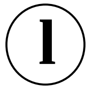
        3. **At "e" place "j", "k" and parent "e"**

    3. **At "b" place "d", "e-subtree" and parent "b".**

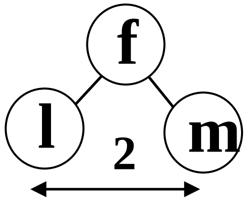
       

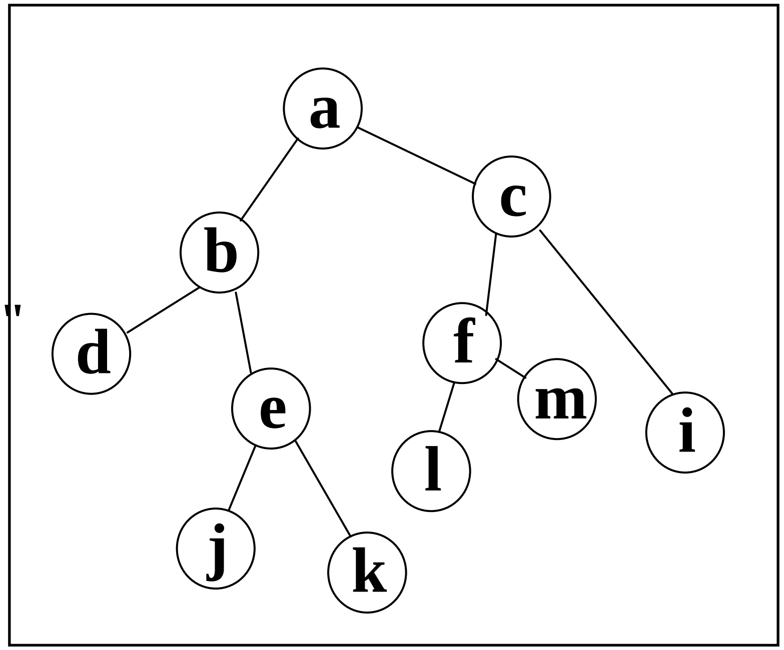


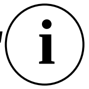
2. **At "a" draw "c-subtree":**
    1. **At "c" draw "f-subtree":**
        1. **At "f" draw "l"** 

        2. **At "f" draw "m"**

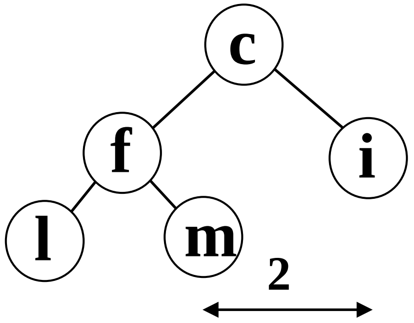
        3. **At "f" place "l", "m" and parent "f"**



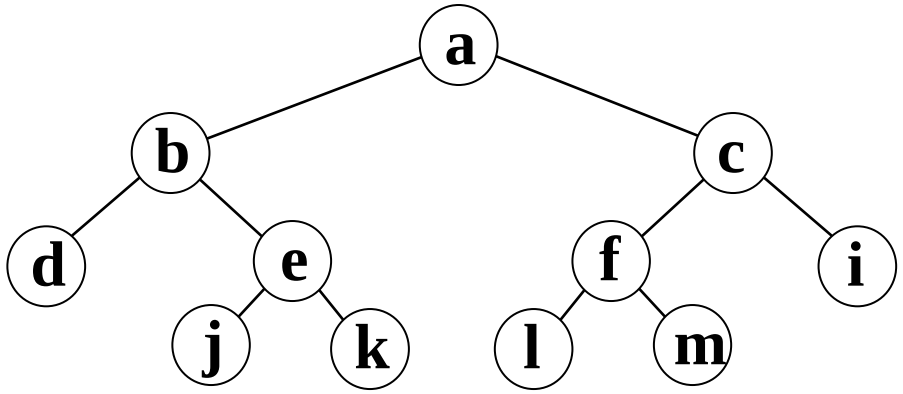


    2. **At "c" draw "i"** 

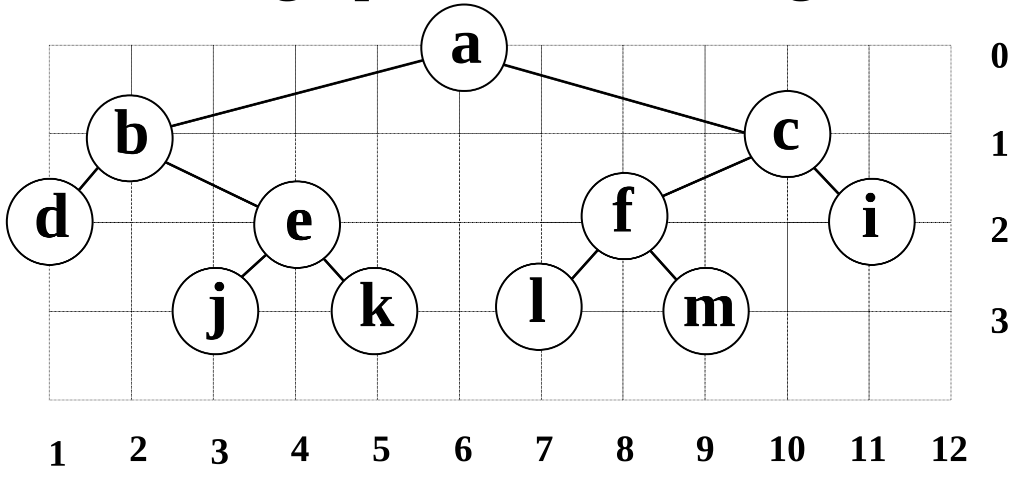
    3. **At "c" place "f-subtree", "I" and parent "c"**



3. At "a" place "b-subtree", "c-subtree" and parent "a"



The same graph drawn with algorithm 1 (layering)



Properties of the divide and conquer approach?

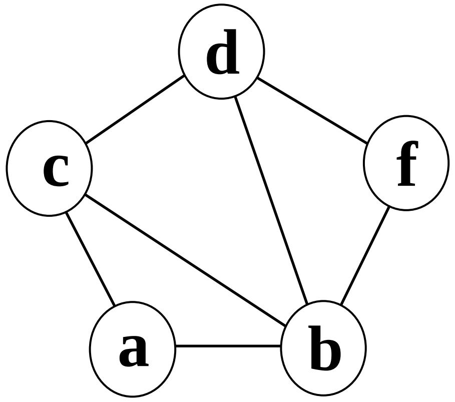
Further graph drawing

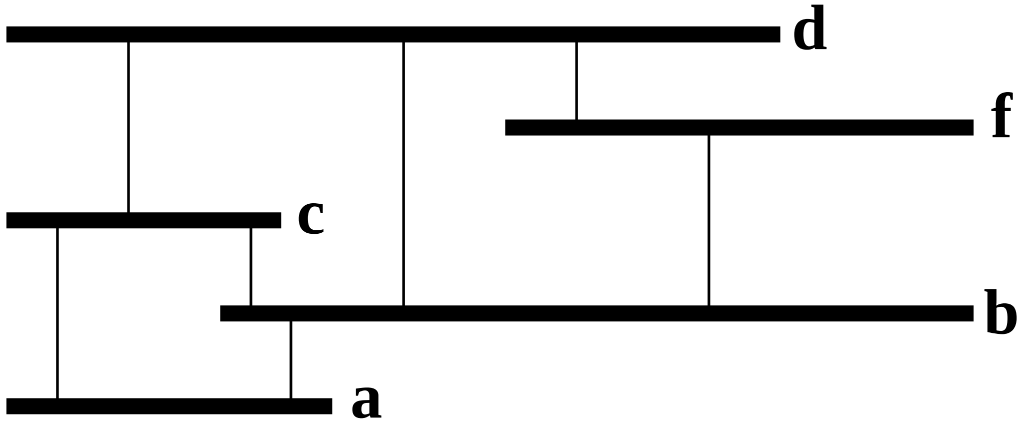
- review motivations for graph drawing
- review requirements of graph drawing
- extend simple rooted tree drawing algorithms to graphs
- address 2 further classes of graph drawing algorithm:
  - visibility approach (for polyline drawing of planar graphs)
  - force-based approaches
- characterise advantages and disadvantages of the algorithms

Visibility representation

The visibility representation Γ of (a planar st-graph) G draws each vertex v as a horizontal segment, called vertex-segment Γ(v), and each edge (u,v) as a vertical segment, called edge-segment Γ(u,v), such that:

- the vertex-segment do not overlap
- the edge-segments do not overlap
- edge segments Γ(u,v) has its bottom end-point of Γ(u), its top end-point on Γ(v), and does not intersect any other vertex-segment





Polyline drawings using visibility

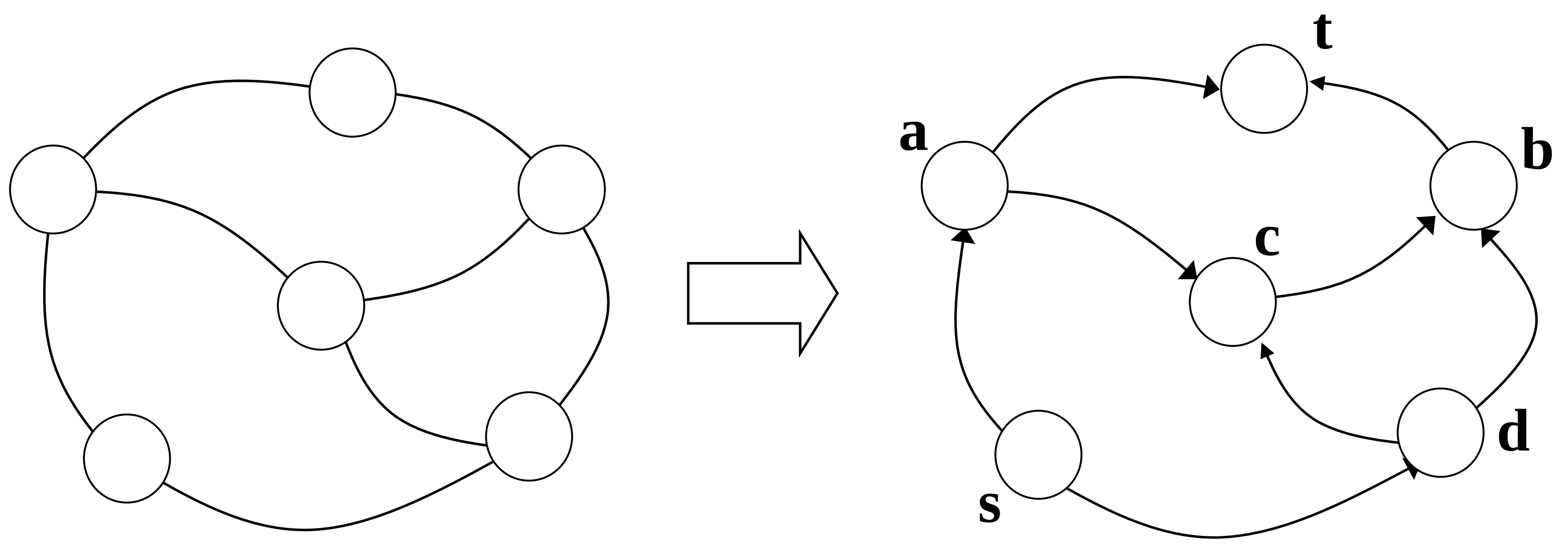
- in drawing a polyline graphs using a visibility representation can be (informally) considered to be a floor plan for the final graph drawing
- however, its construction and use is relatively complex, though it gives some sense of the nature of algorithmic approaches to graph drawing

Polyline drawing using a visibility representation:

1. from a planar graph construct an st-graph, G
2. compute the dual of G, G^*
3. compute the topological numbering of G and G^*
4. identify the orientation of the faces for each vertex
5. construct the visibility representation of G
6. construct the polyline drawing of G

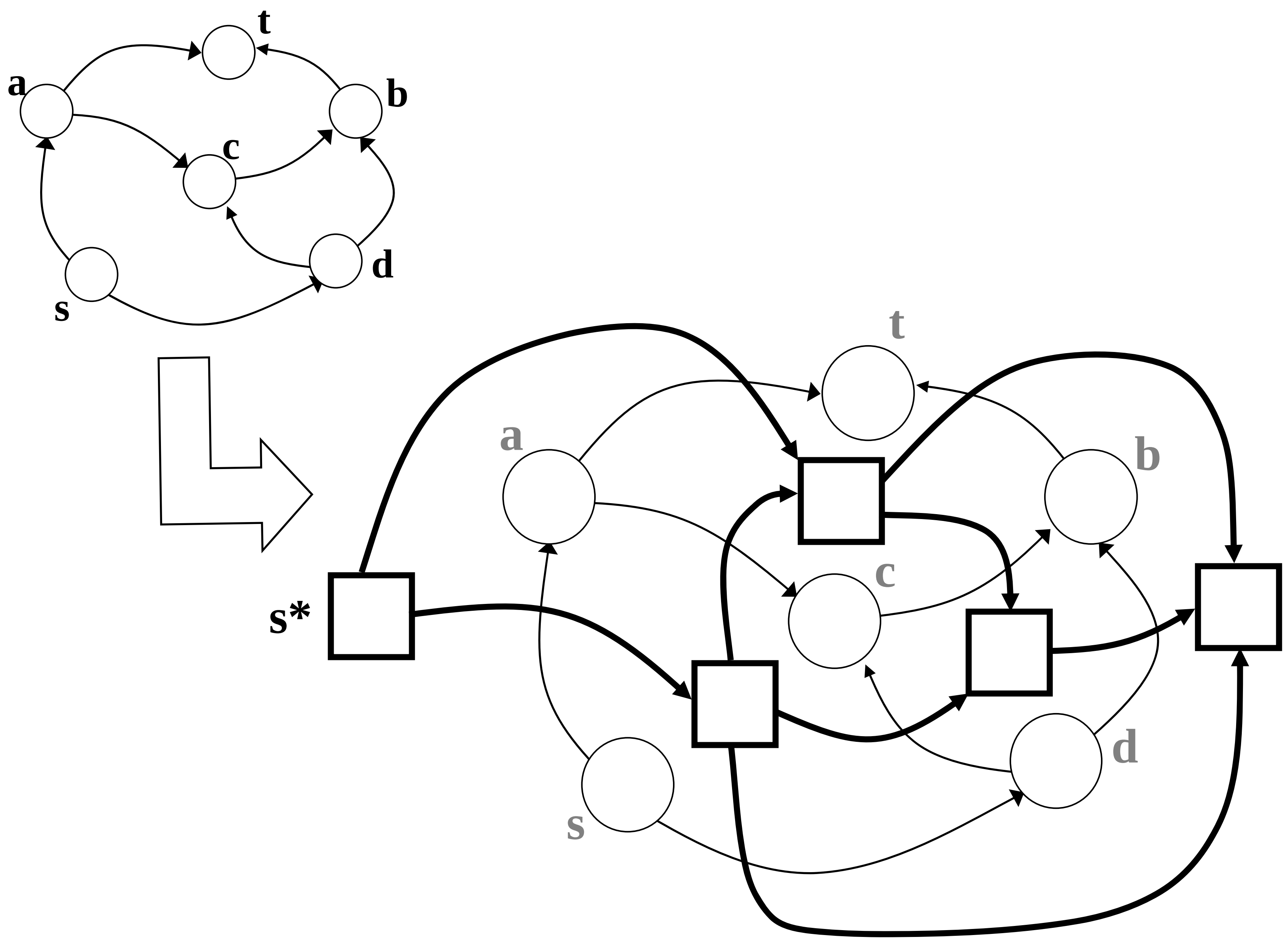
Stage 1: planar st-graph construction

Assuming G is planar, orient G into an st-graph: an acyclic directed graph with a single source s, and a single sink t.



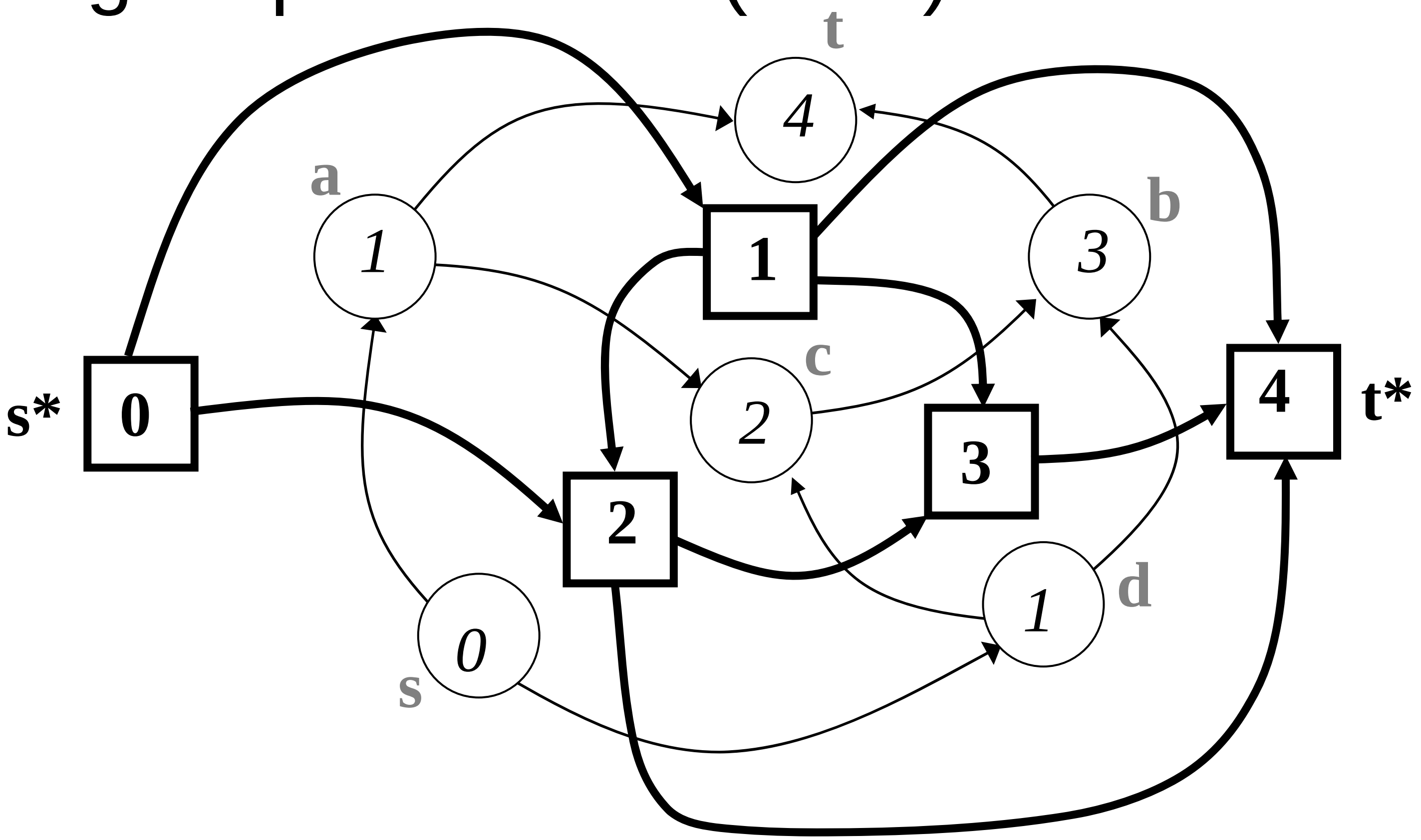
Stage 2: compute the dual of G, G^*

1. Draw a vertex in G* corresponding to each face in G
2. Denote the “left” and “right” external faces of G* as s* & t*
3. An edge exists between two vertices in G* when the two corresponding faces in G share a common edge, except for (s*, t*)
4. For each edge in G, define the “left” face as the one on the left side of the edge (traversing along its direction)
5. Orient the edges in G* "left-to-right"


t
a
b
c
d
s
t
a
b
c
s*
t*
d
s

Stage 3: topological numberings

For both G and G* compute the topological numbering for each vertex: the longest path from s (or s*) to the vertex.

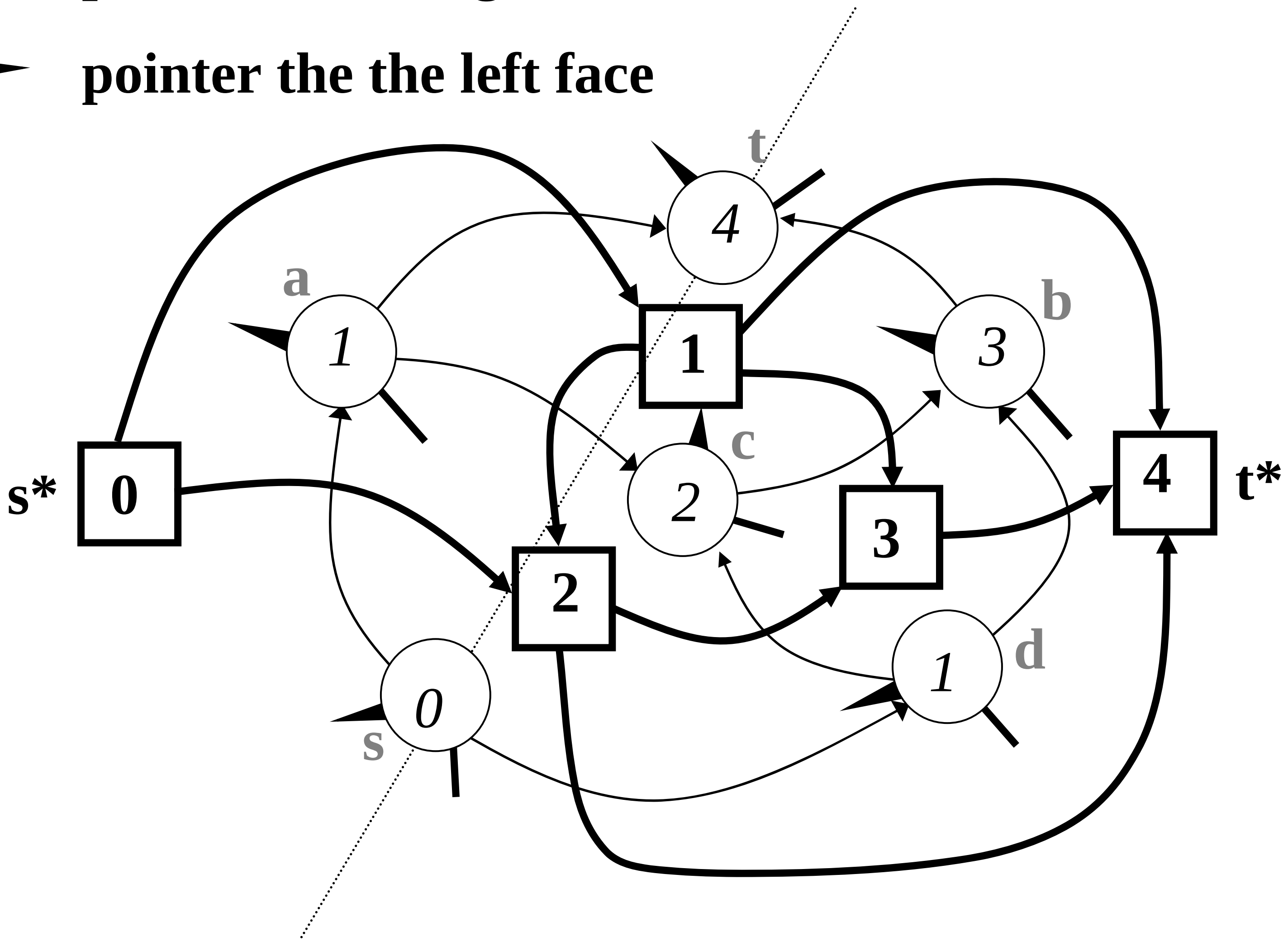


Stage 4: vertex and face orientation

- identify which of the faces are to the left and right of each vertex and each edge
- left(e) means the face to the left of edge e, which means the face to the left as you traverse in the direction of the edge
- left(v) means the face to the left of vertex v, which is the face that separates the incoming from the outgoing edges (clockwise), and right(v) is the face that separates the outgoing from the incoming
- for vertices s and t, the left and right faces are the external faces s* and t*

— pointer to the right face

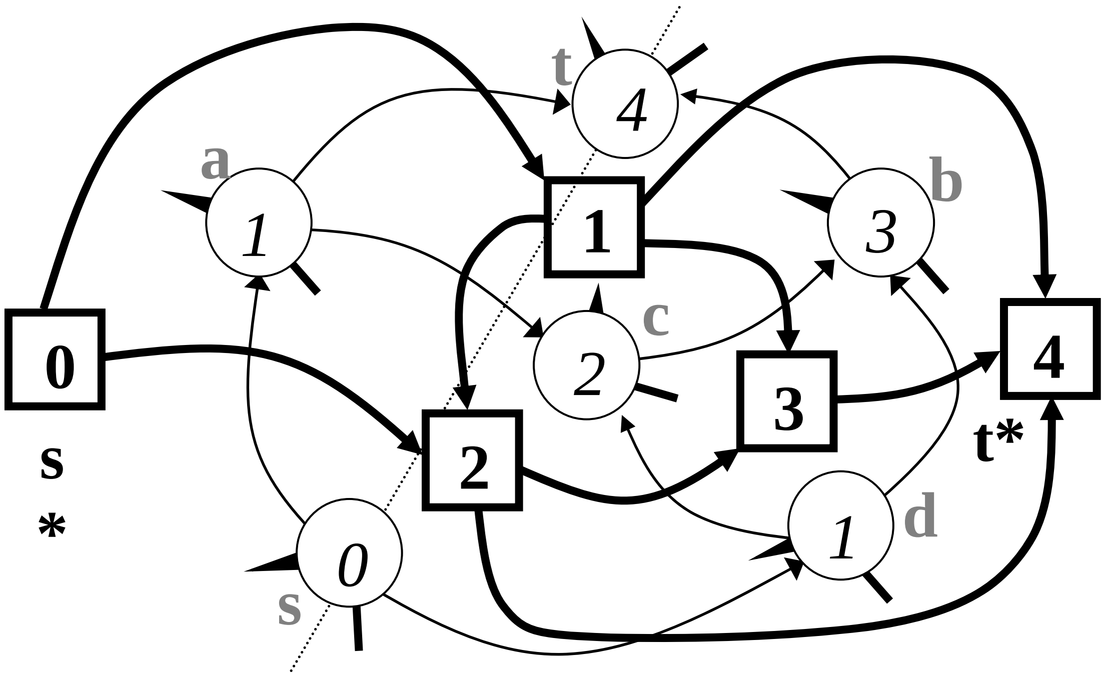
► pointer the the left face



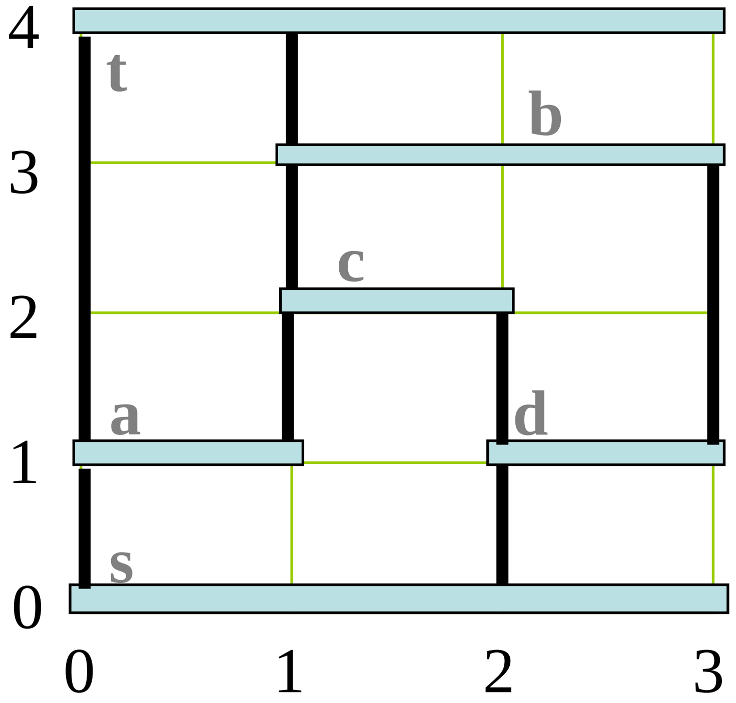
Stage 5: visibility representation

To construct the visibility representation of the st-graph G:

1. Refer to the topological numbering of G as Y, and G* as X
2. For each vertex v, draw the vertex segment Γ(v) at the y-coordinate Y(v) and between x-coordinates X(left(v)) and X(right(v)-1)
3. For each edge e, draw the edge segment Γ(e) at x-coordinate X(left(e)), between y-coordinates Y(orig(e)) and Y(dest(e))



0
s
*
1
a
4
t
1
2
c
2
3
b
3
4
t*
0
s
1
d

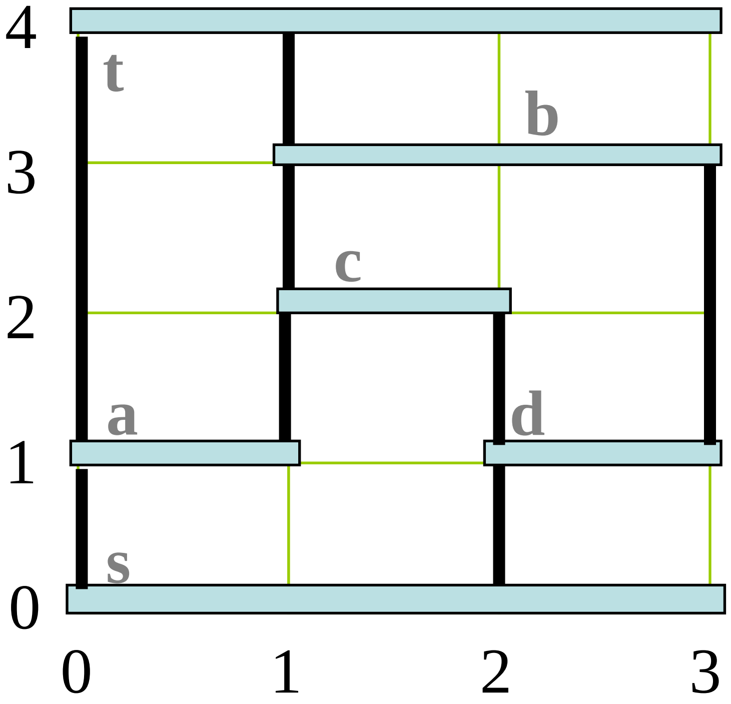


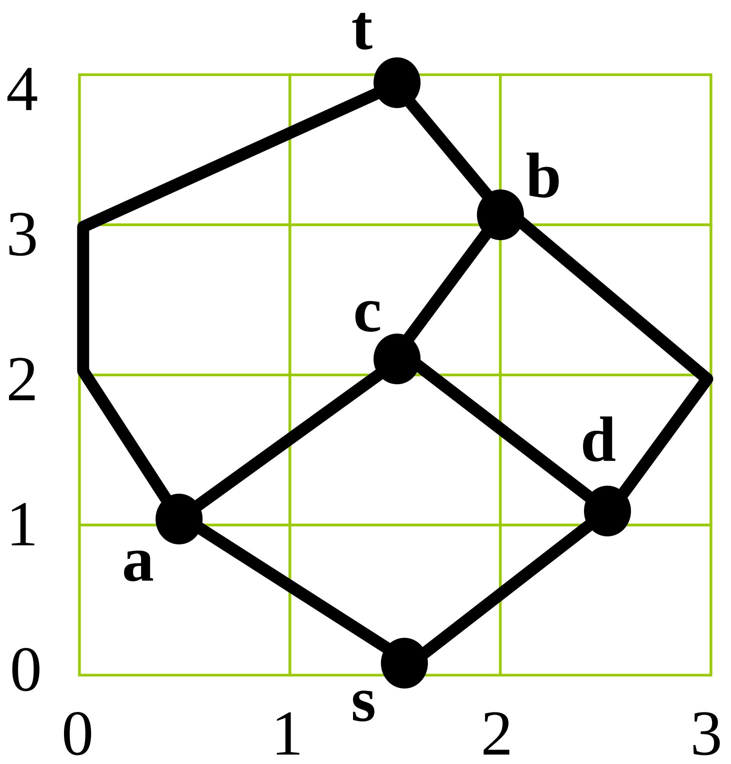
4
3
2
1
0
t
b
c
a
d
s
0
1
2
3

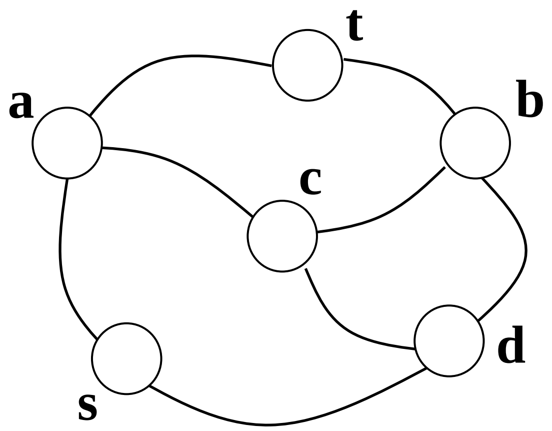
Stage 6: polyline drawing construction

1. For each vertex v, replace the vertex-segment $\Gamma(v)$ with the point $P(v) = (x(v), y(v))$ where $P(v)$ is the midpoint of $\Gamma(v)$
2. For each edge (u,v) do:

```
if $y(v) - y(u) = 1$ then {short edge}
    replace the edge-segment $\Gamma(u,v)$ with a line from $P(u)$ to $P(v)$
else {long edge}
    replace the edge-segment $\Gamma(u,v)$ with a line from $P(u)$ to $P(v)$ passing through $(x(\Gamma(u,v)), y(u)+1)$ and $(x(\Gamma(u,v)), y(v)-1)$
```

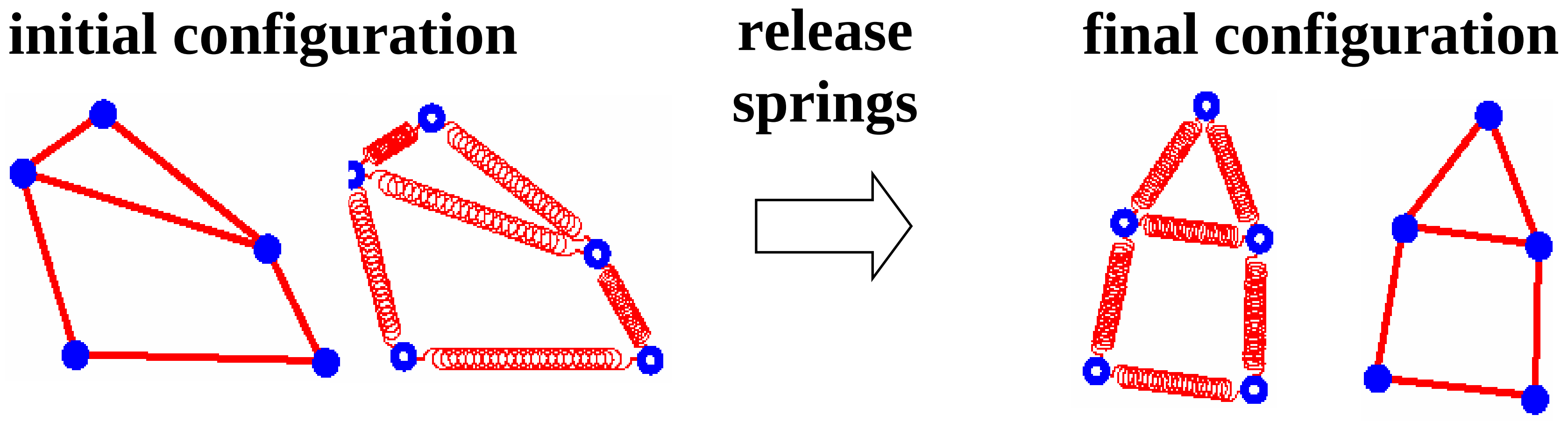

4
3
2
1
0
0
1
2
3
t
b
c
a
d
s


t
b
c
d
a
s
4
3
2
1
0
0
1
2
3


t
a
b
c
d
s

Force-directed methods

- Hirsch formulated a successful solution technique for the PLFP problem using a force analogue, where the repulsion of a label position was estimated from the overlap regions
- graph drawing can also be done by force-directed methods



Framework for force-directed drawing

Force-directed drawing approaches comprise:

- **MODEL**

  A system of attractive and repulsive forces which is mapped onto the edges and vertices of a graph according to the required aesthetic criteria and constraints

- **SOLUTION TECHNIQUE**

  The technique for finding positions for every vertex of the graph such that the system of forces in an equilibrium state (i.e. the sum of the forces at all vertices is zero)

A simple force-directed model

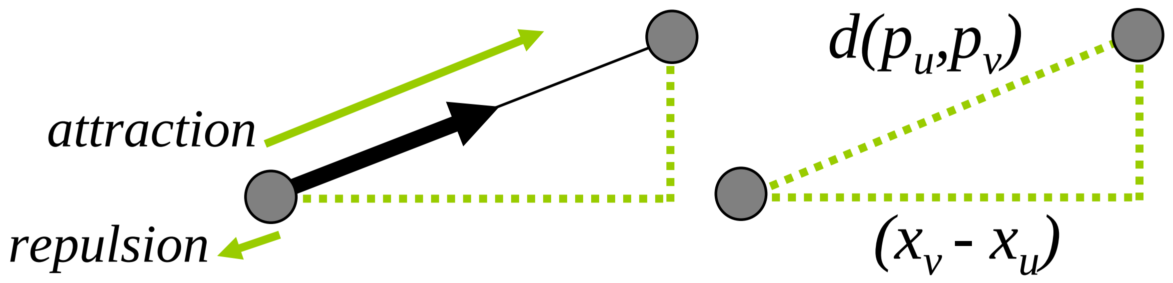
- a simple force-directed model using spring and electrical forces:
  - edges are modelled as springs and vertices as equally charged electrical particles

$$F(v) = \sum_{(u,v)\in E} f_{uv} + \sum_{(u,v)\in V\times V} g_{uv}$$

- force on a vertex, F_{uv} is given by:
  - f_{uv} is proportional to the length of the spring
  - g_{uv} is the electrical repulsion between two particles
- thus, the x-component of the force is given by:

$$F_x(v) = \sum_{(u,v)\in E} k^1_{uv}(d(p_u, p_v) - l_{uv}) \frac{x_v - x_u}{d(p_u, p_v)} + \sum_{(u,v)\in V\times V} \frac{k^2_{uv}}{(d(p_u, p_v))^2} \frac{x_v - x_u}{d(p_u, p_v)}$$

$$F_x(v) = \sum_{(u,v)\in E} k^1_{uv}(d(p_u,p_v) - l_{uv})\frac{x_v - x_u}{d(p_u,p_v)} + \sum_{(u,v)\in V\times V} \frac{k^2_{uv}}{(d(p_u,p_v))^2}\frac{x_v - x_u}{d(p_u,p_v)}$$



- $p_u=(x_u, y_u)$ and $p_v=(x_v, y_v)$ are the positions of vertices u and v
- $d(p_u,p_v)$ is the distance between *u* and *v*
- k^1 is the spring constant, the stiffness of the edge springs
- k^2 is the electrical repulsion constant
- l_{uv} is the unstretched spring length, when $d(p_u,p_v) = l_{uv} \rightarrow$ no force
- $(x_v - x_u) / d(p_u,p_v)$ is the x-component of the repulsion and spring force

Solution techniques and other models

- performing a local search is the simplest solution technique:
  1. randomly place the vertices
  2. loop until all forces are below a threshold:
     - compute the forces on each vertices
     - move each vertex in the direction of, and a distance proportional to, its force
- other force models:
  - barycenter method:
    - no repulsion, l_{uv} to zero for all edges
    - “nail down” the positions of at least three vertices
  - graph-theoretic distances:
    - tries to make the Euclidean distance between all pairs of vertices *u* and *v* (not just directly connected vertices) $d(p_u,p_v)$ proportional to the graph theoretic distance $\delta(u,v)$ (the shortest path length from *u* to *v*)

General energy functions

- simple model: energy function is a continuous function of the locations of the vertices
- many important aesthetic criteria are not continuous:
  - number of crossings
  - number of horizontal and vertical edges
- to optimize several criteria simultaneously they can be linearly combined:

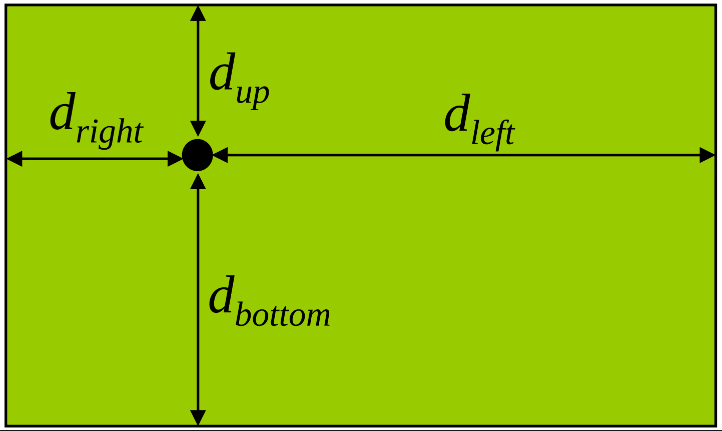
  $\eta = \lambda_1\eta_1 + \lambda_2\eta_2 + \lambda_3\eta_3 + \lambda_4\eta_4 + \ldots$
- for example, (Davidson and Harel, 1996) use the following energy function:

$$\eta_1 = \sum_{u,v \in V} \left(1/d(p_u,p_v)^2\right) \ldots \text{as before}$$

$$\eta_2 = \sum_{(u,v) \in E} (d(p_u, p_v))^2 \quad \ldots \text{as before}$$

$$\eta_3 = the\, number of\ edge crossings$$

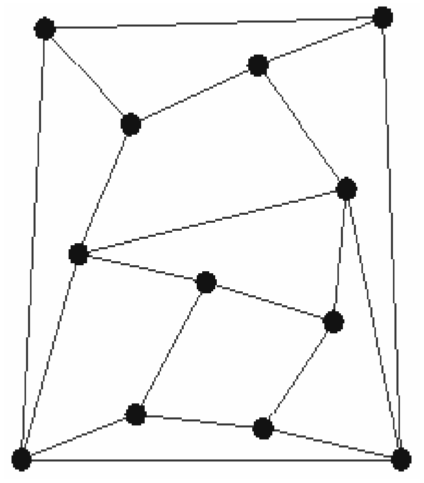
$$\eta_4 = \sum_{u \in V} \left(\frac{1}{d_{left}^2} + \frac{1}{d_{right}^2} + \frac{1}{d_{top}^2} + \frac{1}{d_{bottom}^2} \right)$$

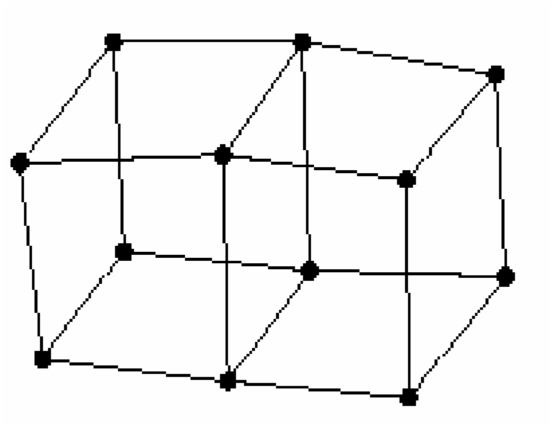


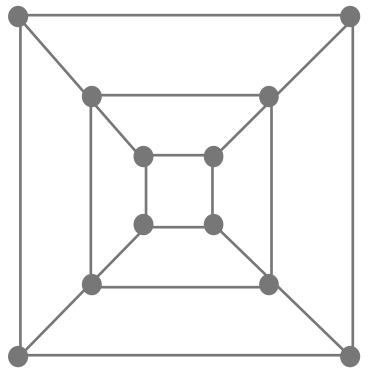
Simulated annealing

- general energy functions require more global optimization techniques, for example, (Davidson and Harel, 1996) used simulated annealing:
  1. choose an initial configuration
  2. repeat some fixed number of times:
     - choose a new configuration C* from the neighbourhood of C
     - if ($\eta^* < \eta$) set C to C*, else set C to C* with a probablity $e^{(E-E^*)/T}$
  3. decrease the temperature T
  4. if the termination rule is satisfied then stop, else continue from step 2
- configuration choice was by random placement of a single vertex within a (decreasing) radius from its current position, thus there is a cooling schedule applied to the perturbation of vertices
- Contrast this with the Christensen’s application of SA to PFLP…

(Davidson and Harel, 1996)


Crossings heavily penalised


Crossings allowed


Symmetric crossing-free drawing not produced

Evaluation of force-based methods

Strengths

- relatively simple to implement
- 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
- Smooth transitions come for free

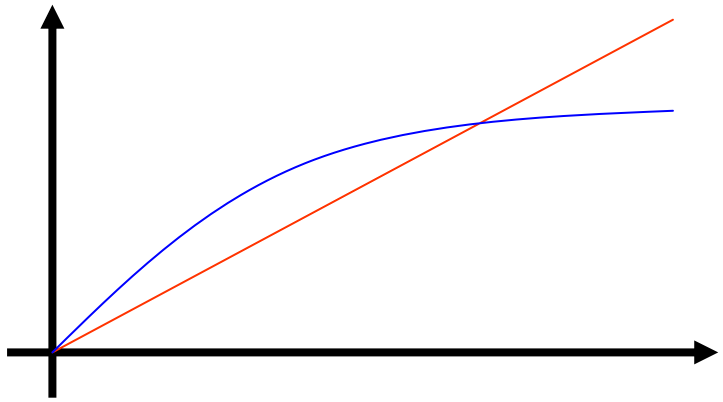
Weaknesses

- slow running time (depending on solution technique, but in general only up to hundreds of vertices)
- few theoretical results on the quality of the drawings produced
- difficult to extend to orthogonal and polyline drawings
- limited ability to add constraints and maintain constraints

Anytime-algorithms

- Provide a result after arbitrary running time
- Result quality increasing monotonously
  - Measurement function for result quality
  - Constant improvement of current results
  - Mostly incremental approaches
  - interruptability
- Can be applied to different base algorithms

- Quality / Time tradeoff



Literatur, Links

- http://www.jgraph.com/
- http://jheer.org/vizster/
  - Movie