1.2 Graph Drawing Techniques

Graph drawing is the automated layout of graphs
We shall overview a number of graph drawing techniques

For general graphs:
Force Directed
  Spring Embedder
  Barycentre based
Multicriteria optimization

For specific graph types:
Planar
Hierarchical
Orthogonal
Graph Drawing is Application Specific

Keep the graph theoretic structure of the graph and map the following applications to the graph:

1. As a UML diagram with 1: 'Person', 2: 'Teacher', 3: 'Class', 4: 'Student'. Person is a generalization of both Teacher and Student. The edges connecting Class to Teacher and Student are associations.

Redraw the graph in a good way for each application.
Force Directed Graph Drawing Methods

A design for layout of graph data structures
Eades’ Spring Embedder


Here we will look at force directed approaches in general and overview of energy systems
Principles of Spring Embedding

A heuristic approach

We need to calculate
1. An attractive force on each vertex, treating edges as springs, forcing the vertices together
2. A repulsive force on each vertex, treating vertices as charged particles and calculated from distance to each other vertex

Then we move each vertex with the balance of the total force
Algorithm

SPRING(G:graph)
place vertices of G in random locations
repeat M times
  calculate the force on each vertex
  move the vertices f*(force each vertex)
draw graph

If d is distance between two vertices
Attractive force = -k*d
Repulsive force = r/d^2

Constants k, r, f, M need to be set by the implementer
(typically through trial and error)
Notes on the algorithm

Each force calculation is the sum of each edge connection and every other vertex. Consider it a vector - it needs direction + amount

Time complexity $O(M \times N^2)$ because calculating distance between all vertices is $N^2$

Hookes law (linear force) used for edges, but Eades originally used a logarithm

Inverse square force for vertices - less repulsion when further apart

Why have most implementations moved from a logarithmic edge force to a linear one?
How the forces work

Length and direction of arrows indicate the force on a vertex. Here for only 2 vertices.
Exercise

Task

Estimate the movement of vertices in this graph for one iteration of a spring embedder. Guess a final layout after 100 iterations.
Generalizing force directed approaches

A force directed graph drawing system consists of:

1. **Model**: a force system calculating force based on the vertices and edges
2. **Algorithm**: a method for finding the equilibrium state of the force system, i.e. where the total force on each vertex is 0

The trick is to find a force system that is both quick to calculate and forms a good graph layout
Other force directed methods

Barycenter based approach
force moves vertices towards their barycenter, the average position of neighbours

Force based on a graph theoretic notions
attempt to get distance between vertices proportional to graph theoretic distance

Magnetic field ideas
parallel, radial and concentric fields

Energy functions that use aesthetic criteria
Barycentre example

Several iterations of moving vertices to the average position of their neighbours

Needs some fixed vertices or all the vertices will end up at the same position

Works well for quick drawing of very symmetric graphs (and that have got obvious fixed vertices)
Pros and cons of force directed methods

Pros
Works well on sparse, smallish graphs
Easy to understand due to force analogy
Quick to implement
Can be used effectively with other methods

Cons
Different results with the same graph
Use with straight line, connected graphs only
Indirect definition of good aesthetics
Multicriteria optimization for graph drawing

Graph Drawing is just another optimization problem

Hence if we can measure the quality of a layout, we should be able to apply standard search mechanisms to improve the quality of the layout. Current search methods used:

- Simulated Annealing
- Hill Climbing
- Genetic Algorithms
Aesthetic metrics for graphs

- edge crossing (total)
- edge length (total, or variance)
- edge bends (total)
- graph size/aspect ratio (various ratios possible)
- vertex separation (e.g. variance of distance to nearest neighbour)
- angular resolution
  - This attempts to avoid very small angles
  - Measured by variance of inverse angle
  - Threshold often used

Application specific measures can be defined.
Putting the multicriteria optimizer together

Measures are combined in a weighted sum. The weights are used to normalize the measures, and define priorities.

In a simulated annealing approach, successive alterations are made to the graph (for instance, random vertex movement) and improvements to the fitness mean the change is kept.

In an effort to avoid local minima, some bad moves are also retained.

Criteria are not orthogonal and so care needs to be taken when assigning weights.
Planar graph layout

Some graphs can be guaranteed to be laid out without edge crossings.
There are fast algorithms (linear time) to detect planar graphs and generate plane layouts.

Interestingly, the problem of finding the minimum number of crossings for a non-planar graph is NP-Complete.

However, plane layouts often need improvement, through e.g. Force directed methods.
Often a planar layout algorithm relies on numbering vertices between a source and target.
Hierarchical drawing

Polynomial time complexity for graphs without cycles. Typically relies on layout on an integer grid.

First, assign layers to vertices

Simple layer assignment (shortest depth) relies on placing vertices in their first available layer, which can lead to ‘layer bloat’

Second, assign vertices to locations on layers. Various mechanisms are possible:

- The x-barycentre of parents or children vertices
- Minimal crossing assignment
- The method must address conflict resolution

Dummy vertices are used when an edge crosses a level
Hierarchical example

Here we use a left to right approach to layout
Orthogonal layout

Here, edges run only vertically or horizontally in a plane graph
Typical layout of a PCB, and some software engineering diagrams
Various algorithms generate layout in fast time
Using specialist techniques for general graphs

If we have a ‘nearly’ planar graph, or a diagram with a sense of flow, but some cycles, we can:

Planarize a graph by adding new vertices or removing edges, then once laid out return the diagram to the previous topology

Make a graph cycle free by reversing edge direction, again returning the layout to the previous topology
Summary

The Spring Embedder produces a fairly symmetric graph with even, short edge length and an even vertex distribution that is fairly easy to implement.

Multicriteria approaches are more flexible, but typically take longer to run and is more effort to implement.

Hierarchical, Planar and Orthogonal layout methods run quickly, but only on specific graph types.