COMP5048 Final Exam Review

# Week 1 – Introduction

## Visualisation

* Use of computer-supported, interactive, visual representation of scientific data (Scientific Visualisation) or (abstract data (Information Visualisation) to amplify cognition.
* **Scientific** Visualisation e.g. astronomy, chemistry, biochemistry. Molecular modelling.
* **Information** Visualisation: Aim to make pictures of abstract data so that humans can understand, navigate and manipulate the data. E.g. Loss of napoleon army, London metro
* R**aw Data -Transformation-> Data Table -Visual Mapping-> Visual Structures -View Transformation-> iews**
* **Representation**: How to represent data
* **Technique**/Algo/Methods: how to construct such representation automatically using a computer efficiently and effectively.

## Good Visualisation:

* Understand the structure, discover new insight, find abnormal/normal patterns/behaviour, generate/reject hypothesis, reveal hidden truth, predict future.

# Week 2 – Complex Data Visualisation I (10 Marks)

## 1 Multi-dimensional/Multivariate Data

* Multiple attributes: **ordinal**, **nominal**, **categorical**, **image**, timestamp, spatial, relational data.

Visualisation **methods**

* **Parallel** **coordinates** (change ordering to see patterns, not that scalable)
  + Nominal Values (edge crossing, change to different method, radial parallel coordinate)
  + Improvements: ncrease width to allow easier understanding by the user. Overlay extra information (e.g. data points on lines). Bad choice for overly complex data.
* **Multi**-**Dimensional** **Scaling** (e.g. **PCA**, might lose info like distance)
  + Porrect to 2D or 3D. Computes similarity matrix between data using distance matrix. Eigen decomposition using largest eigenvalues.
  + Use because 50 dimensions too large, cannot understand. Reduce dimensions to plot.
* **Glyph** (Data-ink ratio, marks/signs (line, point, area), **visual** **variables**)
  + Position (change in x,y location), e.g. hierarchal.
  + Size (change in length, area or repetition) e.g. represent 100 squares as 1 large square.
  + Shape (change in shape) e.g. different shape for categorical values.
  + Values (shade light to dark). E.g. gradient in the value of the data.
  + Orientation (change pattern direction of mark).
  + Colour (change hue at a given value e.g. large value red, low green)
  + Texture
  + Motion
* Bar chart
* Line chart
* Scatter plot

## 2 Spatial Data

* Data with geometry (map, longitude/latitude)

Visualisation methods

* Geo Visualisation (map)
* Trajectory visualisation (e.g. traffic analysis over a map)

## 3 Temporal/Dynamic Data

* Data with time stamps: changing over time. Display the changes in nurmeric data over time of different categories.

Visualisation methods

* Stream graph (overview, trends, not for precise numbers, stacked area) y is not fixed so determined on order
* Theme river (overview, trends, everything is centered to zero or middle)
* Storyline (overview, dynamic clustering). Display temporal dynamics of social interactions. Convery global and local interactions in the data. Change ordering to min crossings. Can see lines moving from one cluster to another (this is local interactions).

## 4 Relational Data with Constraints

Quality metrics (measure of readability/aesthetics)

* edge crossings (avoid)
* drawing area: width\*height (spread nodes evenly by minimise area for fixed sized nodes ore maximise resolution for fixed size screen)
* symmetry (rotational, axial (reflectional), dihedral symmetry)
* edge length (total/maximum/uniform edge length) (avoid long edges)
* bends (total/maximum/uniform bends, # of bends per edge) (avoid)
* angular resolution (min angle between adjacent edges, crossing angle)
* aspect ratio: ration between width and height

**Optimal Solution:** Minimise (crossings, area, total edge length, #bends) maximise (symmetry and angular resolution) **NP-HARD** due to conflicts between each quality metric.

Drawing Conventions. Straight-line, orthogonal, polyline and upward drawing.

### Tree (Hierarchical Relational Data):

Data structure with no cycles

* rooted tree (distinguished vertex in a tree (root))
* directed edges
* leaf = no child nodes.
* Depth = # edge from root to v.
* Height = max depth
* Ordered tree = noded tree with fixed ordering for children of each vertex.

Binary Tree Traversal:

* Inorder:left subtree-root-right subtree
* Preorder: root-left subtree – right subtree
* Postorder: left subtree – right subtree – root

### Layered tree drawing

* rooted (binary) tree T
* **assign y layer according to the depth**
* compute x distances using algorithm

Algorithm - Tidier Drawing Algorithm (Reingold-Tilford 81)

* Divide: recursively apply the algorithm to draw the left and right subtrees of T.
* Conquer
  + move the drawings of subtrees until their horizontal distance equals 2.
  + place the root r vertically one level above and horizontally half way between its children.
  + If there is only one child, place the root at horizontal distance 1 from the child.

Two traversals:

* Step 1. Postorder traversal
  + For each vertex v, recursively computes the horizontal displacement of the left & right children of v with respect to v.
* Step 2. Preorder traversal
  + Computes x-coordinates of the vertices by accumulating the displacements on the path from each vertex to the root.

Implementing Postorder Traversal in Linear Time

Left (Right) Contour:

* sequence of vertices vi, the Leftmost (Rightmost) vertex of T at level i.
* Invariant: store Left (Right) contours in a linked list for each v.

Compute Contours of v:

* scan the Right Contour of the Left Subtree, and the Left Contour of the Right Subtree (follow the linked list).
* accumulate displacements of vertices on the contours
* keep track of the maximum cumulative displacement at each level.

Runtime: O(n)

* It is necessary to travel down the contours of two subtrees T’ and T’’ only as far as the height of the subtree of lesser height.
* the time spent processing vertex v is proportional to the minimum heights of T’ and T”.
* The sum over all vertices v of the minimum height of the subtrees of v is no more than the number of vertices of the tree.

Properties

Constructs a drawing of a binary tree T in linear time such that the drawing is

* layered, planar, straight-line and strictly downward
* O(n2) area
* two vertices are at horizontal & vertical distance at least 1
* parent vertex is centered with respect to its children
* isomorphic subtrees have congruent drawing up to a translation
* axially isomorphic subtrees have congruent drawings, up to a translation & a reflection in y-axis
* requires a lot of area so dendrogram might be better sometimes.

### 2 Radial Tree Drawing

Algorithm

* Layers are represented as **concentric** **circles** Ci.
* Place the **root** (or **center**) of the tree at the **origin**.
* Draw each subtree in **annulus** **wedge** W (**Divide** and **conquer**)
* **Angle** of wedge: **proportional** to **#** of **leaves** of each subtree

How to guarantee planarity? : no edge crossing

* **draw** a **subtree** inside a **convex** **subset** F of the wedge W.
* **Tangent** to **v** on **Ci** meets at a, b on C(i+1)
* F: unbounded region formed by the line segment ab and rays oa & ob

Runtime: O(n) divide and conquer

Properties

* used for free trees
* select the center as a root
* Planar drawing with straight-line edges
* can display symmetry

### 3 HV Tree Drawing

HV is a binary tree where a child of u is either horizontally aligned to the right of u or vertically below u.

Algorithm

Divide:

* recursively construct hv-drawings for the left & right subtrees

Conquer:

* perform either horizontal combination or a vertical combination (place the subtree with the largest number of vertices to the right of the other one for right heavy)
* The height & width are each at most n-1: O(n2) area (Height is at most logn, width is at most n-1: area O(nlogn) for right heavy(place sub tree with largest number of verticies to the right))

Properties

Algorithm Right-Heavy-HV-Tree-Draw construct a drawing of binary tree T with n vertices in O(n) time such that the drawing is

* hv-drawing: downward, planar, grid, straight-line and orthogonal
* area O(nlogn): width is at most n-1, height is at most logn
* simply and axially isomorphic subtrees have congruent drawings, up to a translation
* for each u, a child of u is either horizontally aligned to the right of u, or vertically aligned below u
* bounding rectangles of the subtrees of u do not intersect
* Good area bound, but bad aspect ratio

Better aspect ratio

* use both combinations alternatively
  + horizontal combination: odd depth (height sqrt(n))
  + vertical combination: even depth
* Complete binary tree: O(n) area and constant aspect ratio
* HV drawing of a General binary tree is O(n2) using dynamic programming

### 4 Inclusion Tree Drawing

* Display the parent-child relationship by the inclusion between isothetic rectangles.
* Minimization of area (perimeter, width, height)
* NP-hard for general trees – Polynomial time algorithm for balanced trees
* used for compounds graphs (union of a graph and a tree)
* allow better fit the drawing in a prescribed region

Tree Visualiation methods

* Dendrogram (bended orthogonal edges, leaves in same layer, good for large trees in small area) Hierarchal Clustering
* Radial dendrogram
* Treemap (space efficient, difficult to follow parent child relationship, Cushion Treemap (use shading to identify levels), Voronoi (O(nlogn), treemap in a shape). No explicit edges so can use whole area.
* Sunburst (radial treemap)
* Space filling Circle packing
* Space filling tree (recursive partition)
* Hyperbolic tree browser (fish eye interaction, 3d)
* Balloon tree (radial variation)
* Cone tree (3d, occlusion)
* Polyplane (divide and conquer, platonic solids)
* Collapsible (telescope, only one path at a time)

# Week 3 – Visualisation of Network Data (20 Marks)

## Force directed methods (spring algorithm)

Most popular method for undirected graphs.

1. Place forces between pairs of vertices

* Attraction force: spring that connect particles for each edge
  + Hooke’s law proportional to the difference between the distance between u and v and the zero-energy length of the spring. i.e. edges want to be same length.
* Repulsion force: equal electrically charged particles that repel each other for each vertex pair
  + Inverse square law

2. Find a zero force configuration (equilibrium).

* locally minimal energy
* the sum of the forces on each body is zero.
* equilibrium configuration (sum of force on verticies = 0)

## Variations of Force-Directed Methods:

## 1 Spring & Electrical force.

**Edge**-modelled as spring. **Vertex** equally charged particles which repel each other.

Aim of the force model design:

* **Spring** force: Ensure the distance between adjacent vertices u and v is approximately equal to natural (zero energy) length. i.e. if spring has no force exerted then lu,v = 0
* **Electrical** force: Ensure vertices not too close to each other.

**Spring** **embedder** [Eades 84]

* Uses logarithmic spring force (Hooks law too strong)
* Two aesthetic criteria
  + Uniform edge length
  + Display symmetry

**Simple** **algorithm** “Follow your nose” to find **equilibrium** **configuration**(minimum energy)

* Initially at random location
* At each iteration:
  + Force F(v) on each vertex is computed (each pair)
  + Each vertex v is moved in the direction of F(v) by a small amount proportional to the magnitude of F(v)
* Stops when equilibrium is achieved or some conditions are met.

**Run time: O(n2)**

Advantages:

* Can be relatively simple & easy to implement
* Good flexibility; supports some constraints
* Heuristic improvements easily added
* Smooth animation of the drawing into the final configuration helps preserving the user’s mental map
* Can be easily extended to 3D
* Often able to display symmetries
* Works well in practice for small graphs with regular structure

Disadvantages:

* Runtime can be slow (quadratic time to compute forces)
* Results are acceptable, but not brilliant
* Few theoretical results on the quality of the drawings produced
* Difficult to extend to orthogonal & polyline drawings

## 2 Barycenter method [Tutte]

* Use springs with natural length 0, and attractive force proportional to the length  (no electrical force)
* Pin down the vertices of the external face to form a given convex polygon. E.e. barycenter mapping.

Partition V into two sets:

* fixed vertex (at least 3, nailed down in a convex position)
* free vertex

To achieve equilibrium,

* choose pv so that Fx(v) = 0 for all free vertices;
* similarly, choose pv so that Fy(v) = 0 for all free vertices.
* The equations are linear: The number of equations and the number of unknown variables is both equal to the number of free vertices.
* Solving them equals to placing each free vertex at the barycenter of its neighbors.

Algorithm **Barycenter-Draw**

Input: partition of **V and P**

* V0: at least 3 fixed vertices
* V1: set of free vertices
* Strictly convex polygon P with V0 vertices

Output: position pv

1. Place each vertex u in V0 at a vertex of P and each free vertex at the origin.

2. Repeat

For each free vertex v do force calculation for x and y components until they converge for all free vertices

Run time Solving linear equations:

* Gaussian elimination: O(n3) time
* Fast matrix multiplication method: < O(n2.5) time
* Sparse graphs (planar graphs): Nested dissection method O(n1.5) time
* In practice, the simple Newton-Raphson iteration (Gauss-Seidal or Jacobi iterative method) converges quickly.

Theorem: [Tutte 63]

* G: triconnected planar graphs (3 paths between any pair of vertices)
* f: face in a planar embedding of G
* P : strictly convex planar drawing of f (convex polygon)
* Algorithm Barycenter-Draw with the vertices of f fixed and positioned according to P, yields a convex planar drawing of G.
* Can display of symmetry (uniqueness of the Tutte embedding)
* Poor resolution: exponential area

## 3 Stress minimization method (Force Simulating Graph Theoretic Distance) [Kamada Kawai 89]

Model graph-theoretic distance with **Euclidean distance**

* The forces try to place vertices so that their **geometric** **distance** in the drawing is **proportional** to their graph **theoretic** **distance** (i.e. the number of edges on a shortest path between u and v).
* Choose stiffness parameter k. Springs between vertices that have small graph theoretic distance are stronger. If vertex close to each other then stiffness is larger. Strees is energy on verticies.

Based on Spring embedder model, two criteria

* Minimize the number of edge crossings
* Uniform distribution of vertices & edges

Iterative approach can solve the equation.

* **Choose** a **vertex** that has the **largest** **force** acting on it,
* vertex is moved to a position that minimizes energy (potential energy for the spring), while other vertices remain fixed. Run time is based on computational time to compute shortest path.

Find a local minimum:

* Move to new positions if it reduce the total energy
* Move one vertex at a time: choose a vertex that is in the worst location
* Repeat until it converge (or improvement is less than a threshold)

The algorithm may search for a new local minimum by swapping pair of nodes and then repeat the procedure.

Run time: O(n3)

## 4 Magnetic spring algorithm [Sugiyama, Misue 95]

Variations

* Some or all of the springs are magnetized
* There is a global magnetic field that acts on the spring
* Magnetic field can be used to control the orientation of edges

3 **types** of **magnetic** **fields**

* **Parallel**: all magnetic forces operate in the **same** **direction**
* **Concentric**: the force operates in **concentric** circles
* **Radial**: the forces operate radially **outward** from a point

The three basic magnetic fields can be **combined**

* encourage **orthogonal** **edges** with a combination of **parallel** **forces** in the **horizontal** & **vertical** **directions**

The springs can be magnetized in two ways:

* **Unidirectional**: the spring tends to **align** with **direction** of the magnetic field
  + **spring the force is proportional to Euclidean distance between position of u and v.**
* **Bidirectional**: the spring tends to **align** with the **magnetic** **field**, but in **either** **direction**

A spring may not be magnetized at all

* The magnetic field **induces** a **torsion** or **rotational** **force** on the magnetic springs.
* The magnetic forces are **combined** with the **spring** & **electrical** force.

Algorithm to find equilibrium:

* initially random position and at each iteration move the vertex to lower energy position

Properties

* Can handle directed graphs (unidirectional springs with one of the 3 fields)
  + arcs point downward: downward parallel field
  + Outward: radial field
  + Counterclockwise: concentric field
* Can be applied to orthogonal drawings: combined vertical & horizontal field with bidirectional springs
* Applied with success to mixed graphs (graph with both directed & undirected edges)
* Edges flow in direction of magnetic field.

## 5 General Energy Function

Optimising aesthetics (discrete) with energy function (continuous). Most energy functions are simple continues function of the location of vertices. h = l1 h1 + l2 h2 + l3 h3 + l4 h4 (measure for an aesthetic criterion)

Each h is an aesthetic criterion

* h1: similar to electrical repulsion (vertices do not come too close together)
* h2: Euclidean distance between vertex u and the four side lines of rectangular area (vertices do not come too close to the border of the drawing area)
* h3: edges do not become too long
* h4: the number of edge crossings in the drawing

Many aesthetic criterion are not continuous: # edge crossing, # horizontal and vertical edges, # of bends.

Main problem: computationally expensive to find a minimum energy state (very slow)

* simulated annealing
* genetic algorithm

Flexibility ensures popularity

The model: an energy function that takes into account vertex distribution, edge-lengths, and edge-crossings, and keeps the drawing within a specified rectangle.

The algorithm: simulated annealing

* flexible optimization technique
* very slow; maximum about 30 nodes and 50 edges

Simulated annealing outline **algorithm**:

Temperature = initial temperature;

Randomly position vertices;

Repeat:

For each vertex v

Move v to a random new position;

If the energy increases with this move, then with a certain probability, reject the move;

Decrease Temperature;

Until temperature reaches a lower bound.

Run time: O(|V2||E|)

## 6 Constraints on Force Directed Graphs

### 1 Position constraints

Assign to a vertex a topologically **connected** **region** where the **vertex** should **remain**

* **Single** **point**: a vertex nail down at a specific location
* **Horizontal** **line**: group of vertices arranged on a layer
* **Circle**: set of vertices to be restricted to a distinct region
* e.g. constraints vertices to curves and 3d surfaces (cylinder, cone etc)

### 2 Fixed subgraph constraints

* **Assign** **prescribed** **drawing** to a **subgraph** .
* May be translated or rotated, but not deformed.
* Considering the subgraph as a rigid body.
* For example, **barycenter** **method** is a force-directed method that **constrains** **a set of vertices** (fixed external vertices) to a **polygon**.

### 3 Constraints expressed by forces

* **Orientation** of **directed** **edges**: **magnetic** **spring**
* **Geometric** **clustering** of special set of vertices
* **Alignment** of **vertices**

Visualise graph clustering (clustered graphs)

* For each set C of vertices, add a **dummy** **attractor** vertex vC .
* Add **attractive** **forces** between an attractor vC and each vertex in C.
* Add **repulsive** **forces** between pairs of attractors and between attractors and vertices not in any cluster.
* E.g. add dummy as the centroid of cluster and apply attractive forces between itself and cluster verticies to attract them. Then repulse to other dummy clusters.

## Pros and Cons:

Advantages:

• Can be relatively simple & easy to implement

• Good flexibility; supports some constraints

• Heuristic improvements easily added

• Smooth animation of the drawing into the final configuration helps preserving the user’s mental map

• Often able to display symmetries

• Works well in practice for small graphs with regular structure

Disadvantages:

• Runtime can be slow (quadratic time to compute forces)

• Results are acceptable, but not brilliant

• Few theoretical results on the quality of the drawings produced

• Difficult to extend to orthogonal & polyline drawings

## 7. Fast spring algorithm (FADE)

Spring methods too slow for large graphs therefore FADE

It is feasible to use

1. a spring method, then

2. a geometric clustering method

to obtain a good graph clustering.

**Barnes-Hutt Method** (method of computing forces between stars)

* Use **Quad tree** to **cluster** the stars (Top Left, Top Right, Bottom Left, Bottom Right) Recursively. Each internal node(quadrant) has up to four children (the 4 quadrants). Stop when each quadrant contains one point. 2^d subspace.
* Use the **forces** between the **clusters** to **approximate** the **forces** between **individual** **stars**.
* The contents of a subtree of can be approximated by a mass at the centroid
* The force that a subtree s exerts on the star x can approximate the sum of forces that the nodes in s exert on x. i.e. calculate the force of a star to the next cluster.

FADE

* To compute the force on star x, we proceed from the root toward the leaves.
* At each iteration, the node movements introduced in the previous step improves the quad tree clustering
  + makes the quad tree clustering (geometric clustering) better reflects the graph clustering (combinatorial clustering).

**Algorithm**

1. px = some initial position for each star x;

2. Repeat

2.1 Build the quad tree;

2.2 For each star x ComputeForce(x,root);

2.3 For each star x, px += eFx;

Until px converges for all x;

Where:

ComputeForce(star x; treenode t)

If the approximation is good

then return the approximation;

else return SsComputeForce(x, s), where the sum is over all children s of t.

**Run time: O(n log n)**

Properties

* The Quad tree provides a clustering of the data
* If the data is well clustered, then Barnes-Hutt method runs faster
  + The approximated force is then more accurate
* The spring algorithm tends to cluster the data. This means that we can:
  + Use Barnes-Hutt to compute the clusters as well as the drawing
  + Use the quad tree as the clustering for the clustered graph

# Week 4 – Visualisation of Directed Graphs (10 Marks)

**Sugiyama Method**

Layered networks are often used to represent dependency relations.

* few edge crossings
* edges as straight as possible
* long edges avoided
* edges pointing downward
* nodes spread evenly over the page

The Sugiyama method is useful for

* dependency diagrams
* flow diagrams
* conceptual lattices
* other directed graphs: acyclic or nearly acyclic.

Worst run time O(|V||E|log|E|)

* Each step involved NP-hard problems for optimisation
* There are many good heuristics available for each step

## Step 1 Cycle removal: make acyclic digraph

Input graph may contain cycles

1. make an acyclic digraph by reversing some edges

2. draw the acyclic graphs

3. render the drawing with the reversed edges

Feedback **set**: set of edges R whose **reversing** makes the **digraph** acyclic. Set of leftward edges

Feedback **arc** set: set of edges whose **removal** makes the **digraph** acyclic

**Maximum acyclic subgraph problem:** find a maximum set of E such that G contains no cycles. NP-HARD (max # of E to make G acyclic).

**Feedback arc set problem**: minimum set of E such that G contains no cycles. NP-COMPELTE (min # set to delete)

**Feedback set problem:** Find a vertex sequence with as few leftward edges as possible (arrange in linear order)

### Method 1 Simple heuristics

Depth First Search (or Breadth First Search)

* Compute a linear ordering o of vertices using DFS/BFS
* then delete edges (u,v) with o(u) > o(v)
* poor performance: reverse |E|-|V|-1 edges in worst case
* runs in linear time: O(n)

### Method 2 Greedy Heuristic [Eades 93]

* Source & sink play a special role
  + source: vertex without incoming edges
  + sink: vertex without outgoing edges
  + edges incident to source & sink cannot be part of a cycle
* Successively remove vertices from G
* Add each in turn, to one of two lists Sl & Sr:
  + either the end of Sl or the beginning of Sr

Algorithm:

Intialize both Sl and Sr to be empty

While G not empty do:

While G contains sink do:

Choose sink, u, remove from G and prepend to Sr

While G contains a sourse do:

Choose source, v, remove from G and append to Sl

If G not empty do:

Choose vertex, u, such that the difference outdeg(u) – indeg(u) is maximum, remove from G and appened to Sl

Concatentat Sl and Sr to form S.

Run time: O(n)

## Step 2 Layer assignment: assign y-coordinates

Requirements

1. Layered digraph should be compact: height & width (# verticies in largest layer) & span (h(u) – (h(v))

2. The layering should be proper (no edge has span > 1) : add dummy vertices

3. The number of dummy vertices should be small

* time depends on the total number of vertices
* bends in the final drawing occur only at dummy vertices

### Method 1. Longest path layering (minimize height):

Algorithm

* Place all sinks in layer L1
* Each remaining vertex v is placed in layer Lp+1,  where the longest path from v to a sink has length p

Run Time: linear time using topological ordering [Melhorn 84]

Main drawback: too wide

### Method 2. Layering to minimize width (Coffman-Graham)

Finding a layering with minimum width W subject to minimum height H:

* Precedence-constrained multiprocessor scheduling problem

### Coffman-Graham Layering

* Input: reduced graph G (no transitive edges) and W
* Output: layering of G with width at most W
* Aim: try to ensure the height of the layering is kept small
* Two phases
  + Order the vertices: based on the distance from the source
  + Assign layers: vertices with large distances from the sources will be assigned to layers as close to the bottom as possible
* Width: does not count dummy vertices

1. First phase: lexicographical ordering < S: set of positive integers, max(S): largest element in S .

Labeling vertices

* start from source: label 1
* choose a vertex v: all incoming vertices are labelled
* assign labels to vertices based on lexicographical ordering: form a set S with labels of all incoming neighbours: choose min.

2. Second phase: ensure no layer receive more than W vertices

Assign vertices to each layer

* start from the bottom layer (sink)
* choose a vertex v: all outgoing neighbours have been placed.
* if more than one such vertex, choose the one with the largest label.
* check width w: move to the next layer if the layer becomes full.

Algorithm: **NP-HARD**

Intiate all verticies unlabeld

For I = 1 to |v| do:

Choose unlabeled vertex v, such that label is minimized

K=1, l1= empty, u = empty

While U not qual to V do

Choose vertex such that every vertex neighbour is in U and label is set is maximized

If current width < W and every edge of vertex u is connected to a vertex in lower layer than add to layer.

Else k = k+1, Lk = {vertex v}.

### Method 3. Minimize the number of dummy vertices

* f: # of dummy vertices in a layering defined by y
* f: sum of vertical spans of edges in the layering - # of edges
* **Layer assignment problem**: choose y-coordinates to minimize f
* **Integer linear programming problem:** layering that minimizes the number of dummy vertices can be computed in polynomial time

## Step 3 Crossing reduction: determine the order of vertices in each layer

* Input: proper layered graph
* # of **edge** **crossings** does not **depend** on the precise position of the vertices, but only the **ordering** of the **vertices** within each layer (i.e., combinatorial, not geometric)
* **NP-complete**, even only two layers
* **2-layer crossing minimization problem**
* Heuristics**: Layer-by-layer sweep**
  + At each stage of the sweep, we:
  + hold one layer fixed, and
  + Re-arrange the nodes in the layer above to avoid edge crossings.

## One-sided 2-layer crossing minimization

* Given a fixed ordering of Li-1,
* choose a vertex ordering of Layer Li to minimize # of crossings
* “layer-by-layer sweep”
* from bottom to top
* **optimal solution is NP-HARD**

### Method 1: Barycenter/average (approximation)

* The most popular method
* x-coordinate of each vertex u in L2 is chosen as the barycenter(average) of the x-coordinates of its neighbors
* If two vertices have the same barycenter, then separate them arbitrary by a small amount
* **Run time: O(n)**
* The barycenter method is at **worst O(sqrt(n))** times optimal (since approximation)

### Method 2: Median (approximation)

* x-coordinate of each vertex u in L2 is chosen as the median of the x-coordinates of its neighbors
* if u has no neighbor, then med(u) = 0
* order the vertices in L2: sort L2 on med(u)
* If med(u) = med(v)
  + Place the **odd degree vertex** on the **left** of the even degree vertex
  + If they have the same parity, choose arbitrary
* **Run time: O(n)**
* The median method is **at worst 3 times optimal**, **best is 1.46 times optimal** (since approximation) optimal corssing has atleast xy + 1 whereas meidan has 3xy.

### Method 3: Integer programming (exact) (two-layer crossing minimization problem.)

* Solving integer programs require sophisticated technique**: branch and cut approach** can be used to obtain an optimal solution for digraphs of limited size
* Advantage: find the optimal solution
* Disadvantage: **no guarantee to terminate in polynomial tim**e
* Successful for **small to medium sized digraphs (50 vertices)**

### Method 5: Planarisation

* Use maximal planar subgraph approach**: NP-hard problem**
* Integer linear programming [Mutzel 97]

## Step 4 Horizontal coordinate assignment: assign x-coordinates (Straighten the long edges)

* **Bends** occur at the **dummy** **vertices** in the layering step.
* We want to **reduce** the **angle** of such bends by choosing an x-coordinate for each vertex: without changing the ordering in the crossing reduction step
* Optimization problem with constraints
  + draw each directed path as straight as possible
  + ensure the ordering in each layer (enforce minimal distance)
* may affect the width of the drawing
* Some layered drawing requires exponential area with straight-lines
* Quadratic programming problems can be solved by standard methods, but it requires considerable computational resource
* **O(n) - simple fast heuristic**

# Week 5 – Visualisation of Big Data (10 Marks)

Main approaches to solve the scale problem with big data.

1. Cluster data
2. 2multi-level approach
3. 3d
4. Reduce visual complexity
5. Integration with analysis
6. Integration with interaction.

## Method 1: Cluster the data

Reduces size/dimension

Clustered graph = Graph + tree (underlying graph)

Examples: Hierarchal clustering, K-means, infomap, Louvain etc.

Metrics: Modularity and Coverage.

**Clustered Graph Visualisation**

Criteria:

* Avoid overlap between clusters
* Minimise edge crossing
* Minimise edge-cluster crossings

Simple solution:

* Weighted Spring algorithm with cluster constraints

## Method 2: Multi-level approach (recursive)

Step 1: coarsening step

Step 2: refinement step (replace each node with the cluster it is representing)

Eg.

* Combinatorial clustering: graph partitioning
* GRIP
* Multi-scale Algorithm
* FM3 (Similar to FADE algorithm: Solar merger using Barnes-Hut method)
* K core based multilevel (linear time)
* Infomap to improve multilevel

## Method 3: Use 3 dimensions

* 3d can be drawn without crossing however occlusion problem
* 2.5d is good for temporal and multi relational data
* can always give linear divide and conquer algorithms in trees because no cycles
* can do n log n for multi level

2.5D Network Visualisation Framework

* Trees
* Clustered graphs
* Directed/Hierarchical graphs
* Dynamic/Temporal networks
* Multi-relational networks

e.g.

* Multi-level Representation of Clustered Graph
* MultiPlane Algorithm
  + Divide the large/complex network into clusters
  + Draw each cluster on a 2D manifold in 3D
  + Connect the planes with inter-plane edges
* 2.5D Tree Layout: PolyPlane
* 2.5D Clustered Graph Layout (Weighted 3D Tree Drawing Algorithm: O(n) time + 2D Spring Algorithm: O(n2) time)
* 2.5D Sugiyama Method, 2.5D Scale-free Network Layout

Scale free network

* Exponential Growth
* Preferential attachment

Properties

* Power-law degree distribution
* Sparse, but locally dense
* Small-world property: O(loglogn) average path length
* High clustering coefficient

Examples:

* Webgraph, Social networks, Biological networks

Parallel Plane/Concentric Sphere Layout Criteria

Criteria:

* Minimise occlusion
* Minimise intra-sphere edge crossings
* Minimise total inter-plane edge lengths

## Method 4: Reduce Visual Complexity

* Matrix representation
* Map representation (e.g. Gmap, Voronoi)
  + Treemap is applied on graph rather than tree.
  + Underlying layout for graph is Tutte and Voronoi Diagram on top to define region.
  + Any point within region is the nearest to the centroid of that region.
* Edge bundling method

## Method 5: Integration with Analysis

* Social Network Analysis
* Data Mining
* Machine Learning

Network analysis:

(1) Centrality: important actors/ crucial links

(2) Cohesive subgroups: components, cores, cliques

(3) Structural roles: positions, roles, clusters

(4) Patterns. Compare similarities

(5) Network Statistics/Comparison

### 1 Centrality Analysis

**Degree: local** measure

**Distance measures (global)**

* Betweenness [Freeman 79]
  + low degree can be important (broker, gatekeeper, intermediary)
  + proportion of shortest paths connecting each pair X and Z which pass thru Y
  + O(n3) time
* Closeness: sum of shortest paths to all the other vertices
* Eccentricity: length of the longest shortest path

**Feedback measures**  – Status/Hub/authority/eigenvector

### 2 Cohesive Subgroups

Component:

* Strong/weak component
* Cycles/cyclic component
* Connected/isolated (k-connectivity)
* Cut vertex/ cut pair

k-Core

* maximal subgraph such that in which each vertex is adjacent to at least k other vertices
* vertex degree: >= k
* Identify important dense subgraphs from big complex networks. E.g. k = 6 identify 6 important subgraphs.
* O(n)

(p-q)-Core

* Two-mode network. Biparite graph. (p, q # degree)

Clique : complete subgraph (strong/weak clique)

* n-clique, n-clan, k-plex

### 3 Network Positions and Structural Equivalence

* **Block** **model** (or image matrix) : **reduction** of **complex** **network**
  + **Structural** (exact) Vertices hold identical positions in the network.

If node in block x is connected to node in block y, then every node in block x is connected to every node in block y. Define an edge (x,y) between the blocks and the result is a blockmodel/reduced graph

* + **Automorphism** (same position abstract). Have the same position in a network in abstract sense. Not connected to same exact node but to nodes analogous to the network instead.
  + **Regular** **Equivalence** (same colour). Node have same colour and are connected to pother nodes of exactly the same colour set.
* **Clusters**: cliques, distance, similarity

### 4 Patterns: Triads, Diads, Motifs

* triads
* path
* cycle
* start
* complete

### 5 Network Measures/Statistics

* Degree distribution
* Clustering coefficient
* Diameter
* Average path length
* Connected component
* Density

## Method 6: Integration with Interaction

**Ben** **Schneiderman** mantra:

* “**Overview** first, **Zoom**-in; **Details** on **Demand**”

Types of interaction

* Filtering (brushing)
* Selection
* Zooming/panning
* Collapsing/expanding

Taxanomy

* Select: mark something as interesting
* Explore: show me something interesting
* Reconfigure: show me a different arrangement
* Encode: show me a different representation
* Abstract/Elaborate: show me more or less detail
* Filter: show me something conditionally
* Connect: show me related items

Interaction can help two problems:

* Computational complexity: the layout is only computed for the key frame (a relatively small network)
* Visual complexity: At any one time, only the key frame (a small network) is on the screen

# Week 6 – Visualisation of Complex Data (10 Marks) (Additional from Week 2)

## I Complex Data Visualisation.

## 1 Multi-dimensional/Multivariate Data

* Multiple attributes: ordinal, nominal, categorical, image

Visualisation methods

* (new) **Small** **multiples** (Shows overview/Visual Comparison, **Limitation**: **scalability**)
* Parallel coordinates
* MDS
* Glyph

## 3 Temporal/Dynamic Data

* Data with time stamps: changing over time

Visualisation methods

* (new) Small multiples
* (new) 2.5D Visualisation (stacked graph)
  + union graph (G = G1 + g2 +.. +Gn) Every graph uses Layout D. Global optimisaiton.
  + Change intilisation where layout Di+1 is computed from layout Di. Local optimisition.
* (new) Animation
* Stream graph
* Theme river
* storyline

Criteria for **dynamic** graph/good **animation**

1. **Preserving** **Mental** **Map**. E.g. G at t1 very diff to t2, therefore try to preserve some features to assist visualsiation.

* Orthogonal ordering (Left, Right, Up, Down)
* Cluster: Shape, Proximity
* Topology: Inclusion relationship (inside-outside)

2. Change **Faithfulness**

* Change in Visualisation should be proportional to the change in the Data

3. Memorability: 7 +-2 visualisaitons can remember. Do not change order inbetween animations.

Main apporchaes for Matrix Interpolation.

1. MDS Voronoi Map. (Graph Morphing where weights are changed)
2. Tutte Voroni Map. Graph morphing and Insertion and Deletion operations.
3. Tutte Dual Map. Graph Morphing and Edge Flip (Edges change by insertion and deletion. edges are shared border)

Criteria: Topological Conistency and Smoothness.

## 5 Multi-relational Network

Visualisation methods

* (new) Small multiples (show each combination of relationships)
* (new) 2.5D Visualisation (each plane is a class and edges show relationship)
* (new) Multistory (dynamic multi-relationship with clusters)
* (new) Other (Temporal map, Gmap, Streamgraph)

## Design/Evaluate VA System

Define Task:

* Show the structure
* Discover new knowledge/insight
* Find regular/abnormal patterns
* Confirm/reject hypothesis
* Confirm expected or discover unexpected
* Predict future.

For example:

* Data: IMDB
* Task: Find temporal patterns, outliers, trends groups etc
* Data Processing: extract meta data, collaboration network, co-citation network etc.
* Design
  + Analysis: clustering, centrality analysis etc
  + Visualisation: parallel coordiantes, treeemap, sugiyama (spring algorithm)
* Implement: Hardware, Software, Tools, Languages, GUI
* Evaluation: Case study, survey, Heuristics etc.

1. Overview first, then Details on demand e.g. 2.5D multi level graph (union graph)

2. Overlay analysis using visual variables (data-ink ratio)

3. Integrate a number of analysis and visualisation methods

4. If the data is big/complex, reduce the data set e.g. 2.5d Scale free netowrk

5. Storytelling with the data: narrative visualization

6. Build a hypothesis: Look for visual evidence to confirm/reject the hypothesis

# Week 7 – Perception/Colour (10 Marks)

## 1. Human Perception System (how human understand visualisations)

### Stage 1 Visual Perception System

Visualisation -> Eyes -> Brain:

**Parallel** **processing** of **orientation**, texture, colour and motion features

* Rapid parallel processing: billions of neurons;
* Extraction of orientation, texture, colour, and motion features

**Detection** of **2D** **patterns**, **contours** and regions

* Slower processing than stage 1;

**Object** **Identification**, Working Memory

* **Slow** **serial** **processing**;
* **Involve both working and long-term memory;**
* Object identification and eye-hand coordination.

### Stage 2 Visual Attention and Pre-Attentive Patterns (Pop out effects)

Stage 1 the whole visual field is processed in parallel and very fast. Information in Stage 2 allows easy distinguishability.

Law of Pre-Attentive Display: Must stand out on some simple dimension (does not often work in conjunction)

* Colour (1 colour different) (too many colours BAD)
* Orientation (one in off orientation)
* Motion (one moves but others stay fixed)
* Size (one larger than another)
* Simple shading (depth)
* Conjunction (BAD) e.g. Different sizes and different colours.
* Compound (BAD) e.g. add on small edge to marks. Hard to find since so similar.

Information Visualisations: Avoid large areas of strong colours. Use different channel for different type (categorical)

## 2. Gestalt principles

Our tendency to bring pieces of info into meaningful wholes.

We **perceive** **objects** as well-**organized** **patterns** rather than separate components.

Based on the concept of “grouping”. 3 main principals: Figure/Ground, Grouping and Goodness of figures.

**Figure/ground relationships**

* Elements are perceived as either figures (objects) or ground (the background or landscape on which the figures rest).
* Reversible relationship (not seen simultaneously) depends on what user perceives as figure or ground.

**Grouping:**

* Proximity
  + Things that are close together are perceptually grouped together
* Similarity
  + Similar elements in shape, size, color, proximity, motion, and direction are grouped together
* Continuity
  + People tend to see things as smooth and continuous, rather than ones that contain abrupt changes.
  + Objects arranged in either a straight line or a smooth curve tend to be seen as a unit.
  + The eye follows along a line, curve, or sequence of shapes identifying visual relationships.
* Closure
  + People tend to first look for a single, recognizable pattern.
  + We have a tendency to visually close gaps in a form.
  + The more familiar the form the faster we visually close the gap.
  + Closure occurs because we seek to make the forms stable.
* Common fate
  + Objects that move together are seen as related

**Goodness of figures**

* Humans tend to interpret ambiguous or complex images as simple and complete.
* People tend to perceive things based on the simplest and most stable or complete interpretation

## 3. Colour

**Categorical** Data:

* **Hue**: pure colours (not mixed with B/W)

**Ordered** Data:

* **Saturation**: White mixed with Hue
* Lightness: black mixed with Hue
* **Luminance**: better match with perception than lightness.

Colour deficiency

* Avoid encoding hue alone - redundantly encode
  + vary luminance
  + change shape
* blue orange is safe. Reg/green bad
* colour appear different depending on adjacent colours (e.g. white border make fill look lighter).

Colourmaps

* Categorical vs Ordered (Sequential (1 hue range from min to max), Diverging (two hues at end points and neutral in middle)
* **Continuous** (for **Quantitative** data) vs **Segmented** (for **Categorical** data)
* **Bivariate** colormaps: **encode two attributes simultaneously**
  + (easy if one attribute is binary; difficult if both attributes have multiple levels e.g. divergin with sequential)
* **Small** **regions** need **high** **saturation** (8 colours)/large regions need low saturation (10 colours)
* Saturation/Luminance: 3-4 colours bins max.
* Reduce # of colours by combine other visual encoding, size and shape. E.g. 1 colour per 21 chromosones.

Rainbow colormap: poor default

* Benefits:
  + mid-level structure visible
  + nameable
* Problems:
  + Perceptual nonlinearity
  + Expressivity mismatch of using hue for ordering
  + Accuracy mismatch of using hue for fine-grained detail

Alternatives

* large-scale structure: fewer hues e.g. two hue continuous
* fine structure: multiple hues with monotonically increasing luminance e.g. multi hue continuums
* Segmented Rainbows for categorical data: Perceptually linear (fine for small # of categories)

## Rules of Thumb

* No unjustified3D
  + Power of the plane instead of depth. Easier to acquire infrom on image plane than depth.
  + Disparity of depth
  + Occlusion hides information
  + Perspective distortion dangers (viz dependent on viewing angle of user)
  + Tilted text isn’t legible
* No unjustified2D
* Eyes beat memory
* Resolution over immersion
* Overview first, zoom and filter, details on demand
* Responsiveness is required
* Function first, form next

# Week 8 – Evaluation Methods (5 Marks)

## Quantitative metrics:

1. Quality Metric *Q*. Q(v1) > Q(v2), 0<Q<1.

Readability Metric (Aesthetics):

Measure thje quality of the perception function. How well the human understands the visualisation. Psychologuical concept.

* Crossings
* Area
* Edge Length
* Bends
* Resolution
* Crossing Angle
* Aspect Ratio

Faithfulness Metrics:

Measure of the quality of the visualisaiton function. Measure of how well the visualisation represents the data (ground truth). Mathematical concept. If data is very dense, large and complex, it is still faithful even tho not perfect structure. As long as proper representation.

* Stress (geometric different between vertices to length of graph area)
* Symmetry
* Shape
* Cluster
  + Ajusted Rand Index (ARI): # pairs of in both ground truth clsuters and viz clusters.
  + Fowlkes-Mallow index (FMI): measure between the two clusters using TP, FP and FN.
  + These scores are most sensitive in capturing changes in quality.
  + SFDP (YiFan Hu) best non-cluster focused layout. MDS bad
* Change
* Quality Metric.

Shape-faithfulmetrics (the shape of the drawing should be faithful to the input graph.)

* deformation experiemtn (progressivaley add noise)
* GION untangling experiment

Cluster-faithfulmetrics (adjuster rand index, homogeneity etc)

* perturb
* layout comparison (compare cluster focused layout with force directed)

## Qualitative Evaluation / Usability (HCI Evaluation methods)

Usability measures the quality of a user's experience when interacting with a product or system

* Ease of learning
* Efficiency of use
* Memorability
* Error frequency and severity

## Survey: easiest method to evaluate usability.

* **Interview**

• Qualitative technique

– Gathering information about users by talking directly to them

– A method for discovering facts and opinions of the users.

• Format:

– It is usually done by one interviewer speaking to one user at a time.

– Structured interviews: a pre-defined set of questions and users

– Open-ended interviews: allows for an exploratory approach to uncover unexpected information.

• Problems:

– The **unstructured** **nature** of the resulting data can be easily misinterpreted.

* **Questionnaire**

• **Qualitative** **technique**

– But results can be quantified

• Preparation

– Keep questions simple, be clear and concise

– Group questions appropriately & give explanation

• Pilot questionnaire before distributing it

– It is still unreasonable to think that any one person can anticipate all the potential problems

• Problems

– It is only as good as the questions it contains

Allows statistical inference. E.g. average score on satisfaction.

* **Focused** **group**

• A focus group is a small-group discussion guided by a trained leader (moderator) in an interactive setting.

• It is used to learn more about opinions on a designated topic, or a product, and then to guide future action.

• A focus group:

– is an interview,

– conducted by a trained moderator among a small group of respondents,

– conducted in an unstructured and natural way where respondents are free to give views from any aspect.

• **Pros**

– It is flexible.

– It generates quick results.

– It costs little to conduct.

– Group dynamics often bring out aspects of the topic or reveal information about the subject that may not have been anticipated by the researcher or emerged from individual interviews.

• **Cons**

– The researcher has less control over the session than he or she does in individual interviews.

– Data are often difficult to analyse.

– Moderators require certain skills.

## Analytic inspection:

Experts review system usability. Critiques can be formal or informal reports. Gerneate results quickly and used early in design phase.

### Heuristic Evaluation (Principles, Guidelines)

1. Briefing session to tell experts what to do.
2. Evaluation period of 1-2 hours in which:

– Each expert works separately;

– Some time is used to get a feel for the product;

– Other time is used to focus on specific features.

1. Debriefing session in which experts work together to prioritize problems.

**Advantages**:

• Few ethical & practical issues to consider because users not involved.

• Can also be difficult & expensive to find experts.

• Best experts have knowledge of application domain & users.

**Disadvantages**:

– Important problems may get missed;

– Many trivial problems are often identified;

– Experts have biases; they are not real users.

– How: guide by existing design principles and heuristics

– What: examines if the system in question abides by recognized usability principles

– Why: to see if the system can comfortably be used based on prior experience in similar systems

### Cognitive walkthroughs (Based on task scenarios)

• Focus on ease of learning.

• Designer presents an aspect of the design & usage scenarios.

• Expert is told the assumptions about user population, context of use, task details.

• One or more experts walk through the design prototype with the scenario.

• As the experts work through the scenario they note problems.

– How: guide by asking questions from the perspective of the user.

– What: performs list of specific tasks

– Why: to see if the tasks can be performed in the correct sequence of actions they were designed in

## Empirical evaluation:

### Observational experiment (Observation, problem identification)

• In an observational usability test, representative users try to do typical tasks with the product, while observers, including the development staff, watch, listen, and take notes.

e.g. think aloud, eye tracking

– Formative: helps guide design

– Single UI, early in design process – Few subjects

– Identify usability problems

– Qualitative feedback from users

### Controlled Experiment (Formal controlled scientific experiment, Comparisons, statistical analysis)

• It is a test of the effect of a single variable by changing it while keeping all other variables the same.

• A controlled experiment generally compares the results obtained from an experimental sample against a control sample.

• General terms

– Participant (subject)

– Independent variables (test conditions), dependent variable, – Control variable, random variable

– Confounding variable

– Within subjects vs. between subjects

– Counterbalancing

Process

1. Determine research questions and conduct literature review

2. Form hypotheses and null hypotheses

3. Identify independent variables and levels of them

4. Identify dependent variables

5. Conduct experiments and analyze data

6. Apply statistical methods and determine p values

7. Reject null hypothesis if p<0.05

8. Accept/(reject?) hypothesis

– Summative: measure final result

– Compare multiple UIs

– Many subjects, strict protocol

– Quantitative results, statistical significance

**Practice Questions**

Question 1 (10 Marks)

1. Tidier tree algorithm. Explain with time complexity.
2. HV/AV Tree drawing. Explain with time complexity.

Question 2 (15 Marks)

1. Tatt’s Barycenter method in detail.
2. Explain Coffman-Graham in detail.
3. Explain median method with time complexity.

Question 3 (20 Marks)

1. 3 temporal/dynamic methods for graph.
2. 3 methods of multi-attribute for graph.
3. 3 methods for…

Question 4 (20 Marks)

1. Explain Heuristic and …
2. Cognitive and Observation experiments
3. 3 Gestalt’s principles
4. Why rainbow colour is not good and explain alternative ways.

Question 5 (15 Marks)

1. Explain spring algorithm in detail.
2. Fade algorithm in detail.
3. How spring algorithm can be modified for clustered algorithm

Know:

* main idea
* aim (what problem do they try to solve)
* how data is structured
* optimisation criteria
* mathematical formula
* Algorithm with steps and pseudo code
* Run time
* Quality metrics
* Theories explaining properties of the algorithm