

COMP5048

Visual Analytics

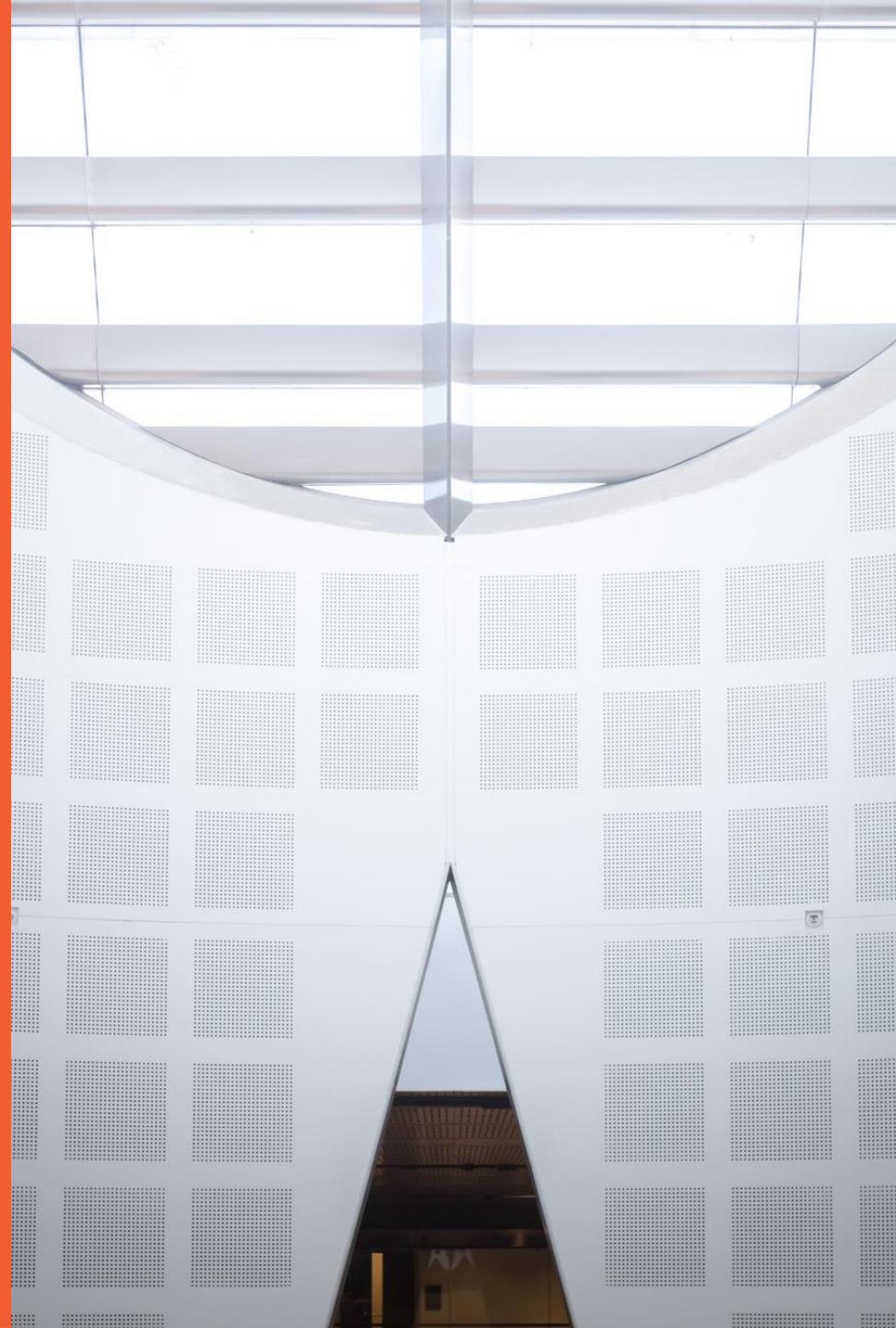
Week 3

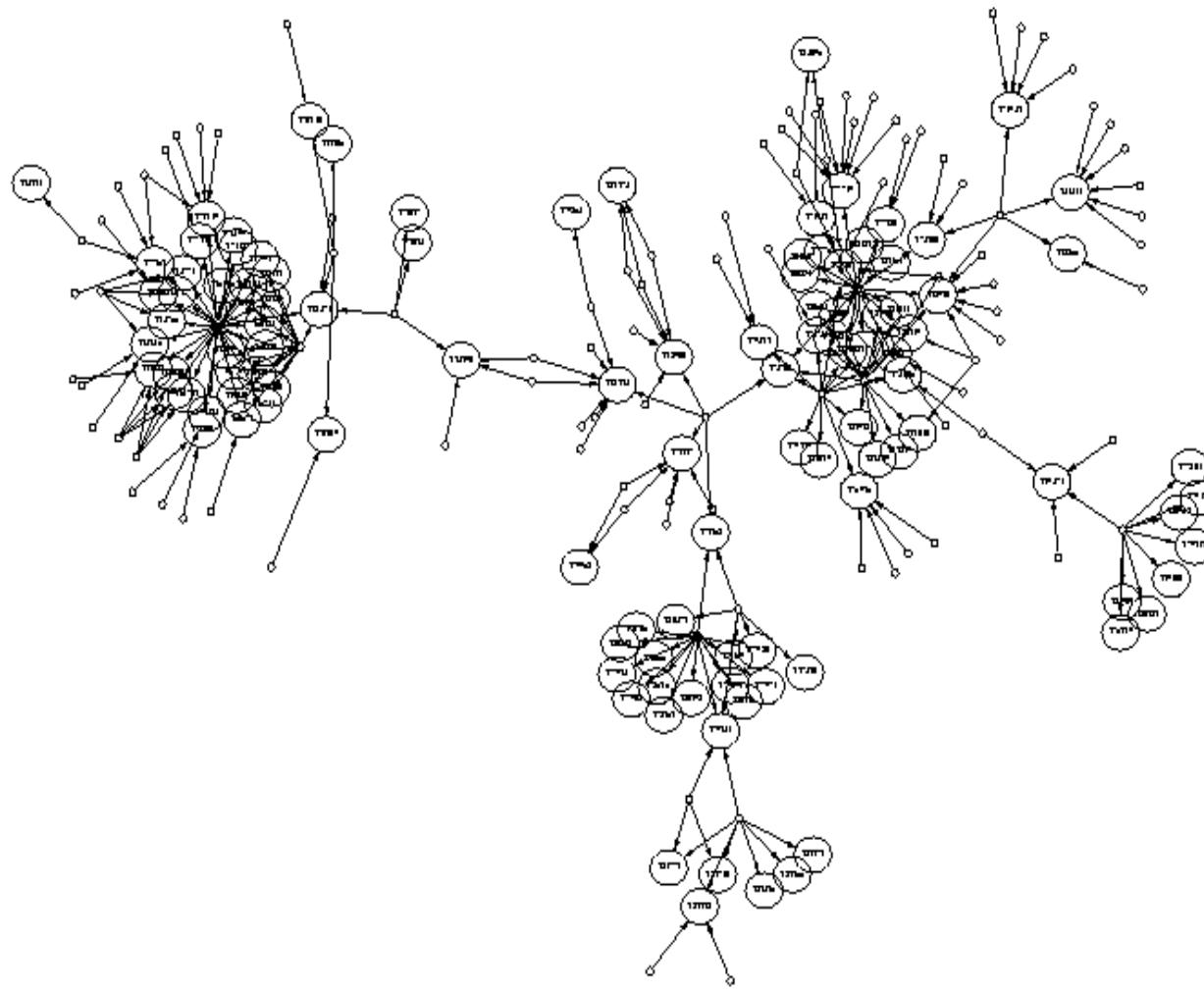
Visualisation of Relational Data I:

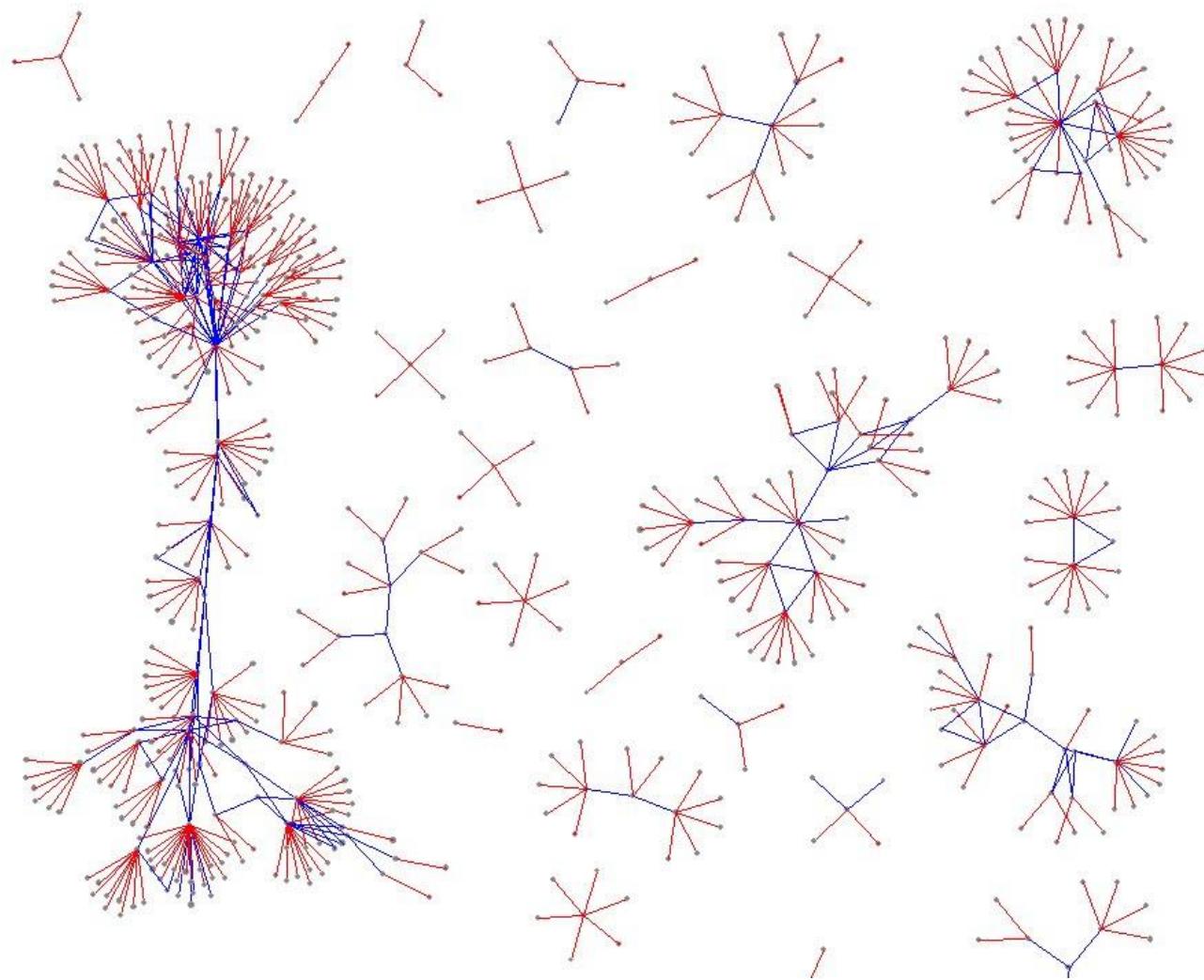
Professor Peter Eades
School of Information Technologies



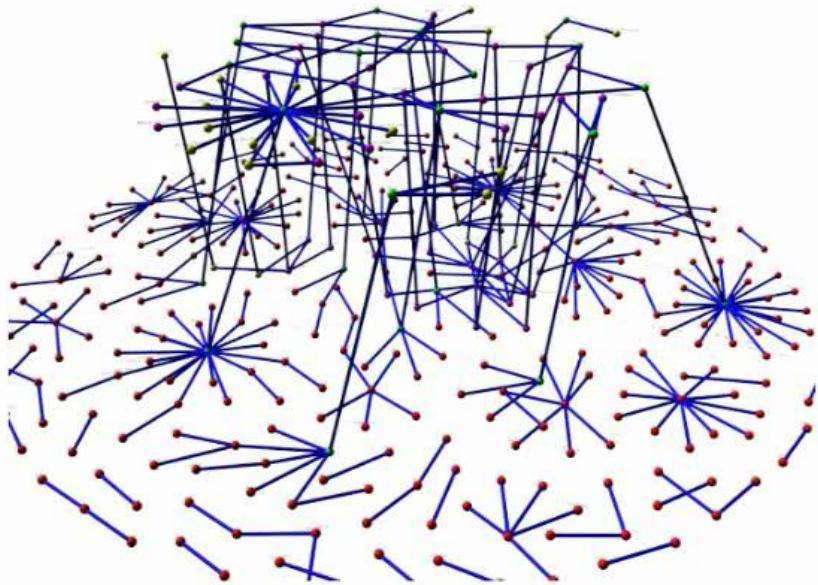
THE UNIVERSITY OF
SYDNEY







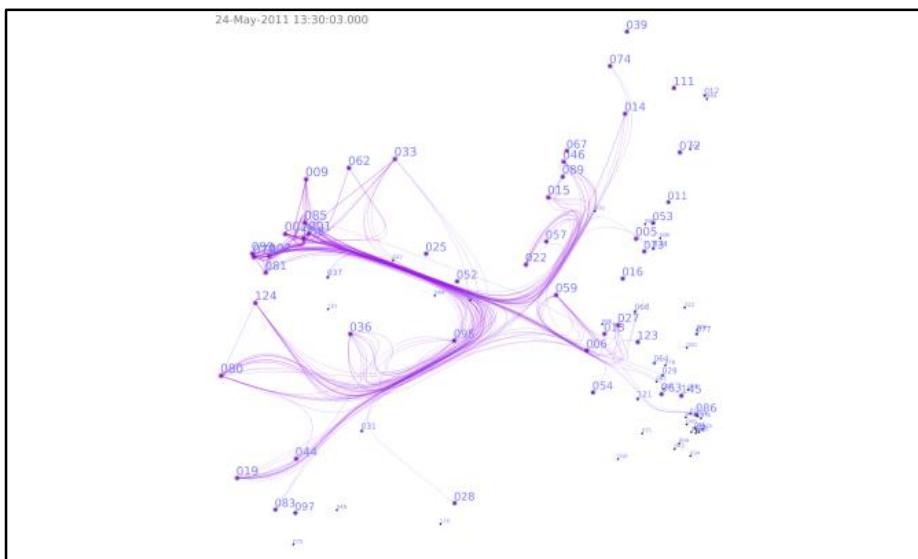
Motifs in a protein-protein interaction network

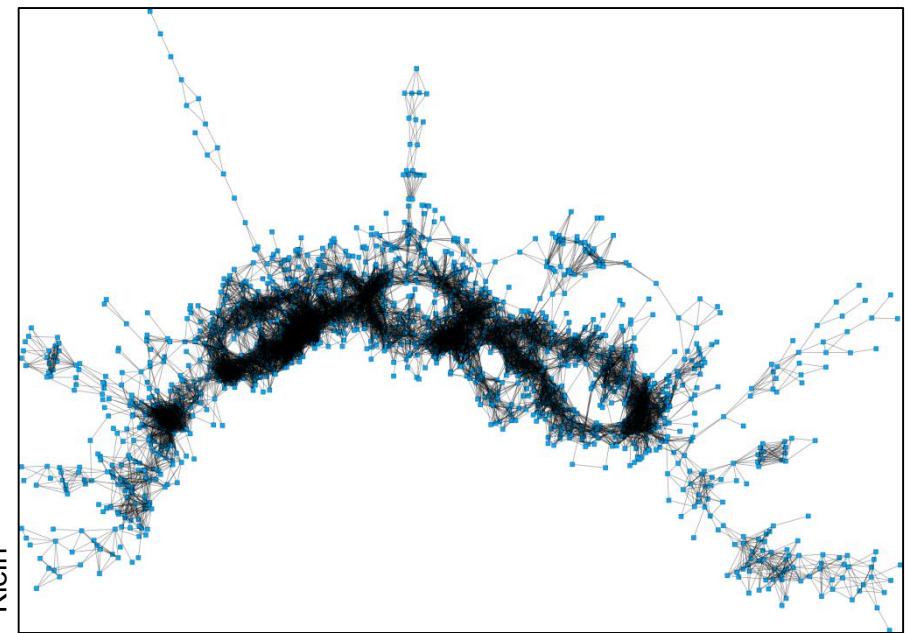


Lanbo Zheng et al.

Klein

Nguyen

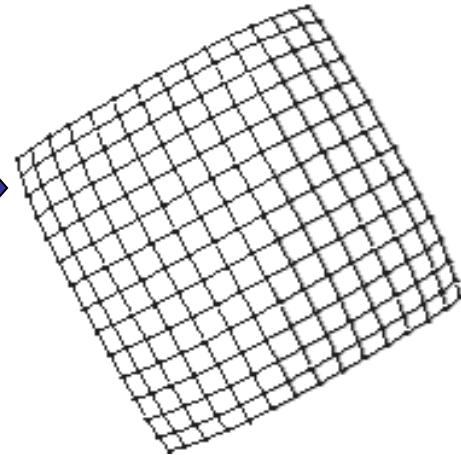
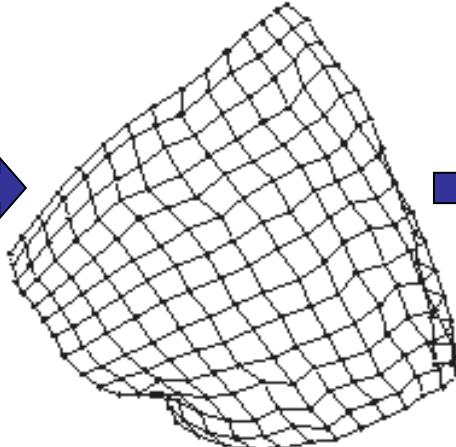
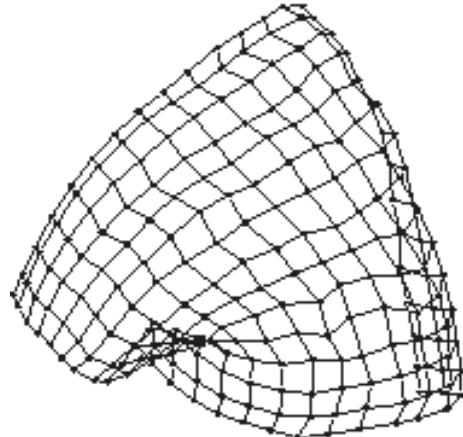
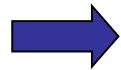
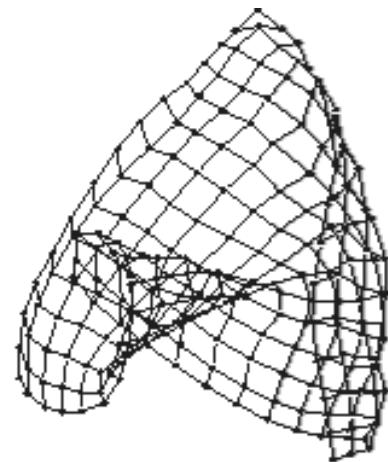
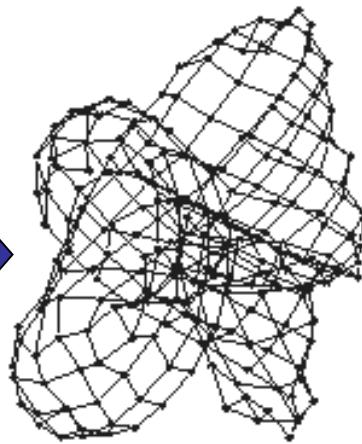
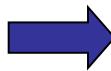
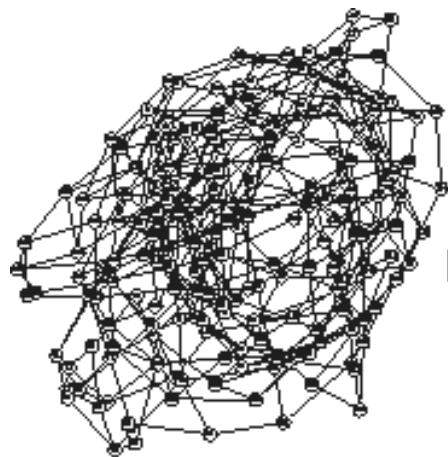


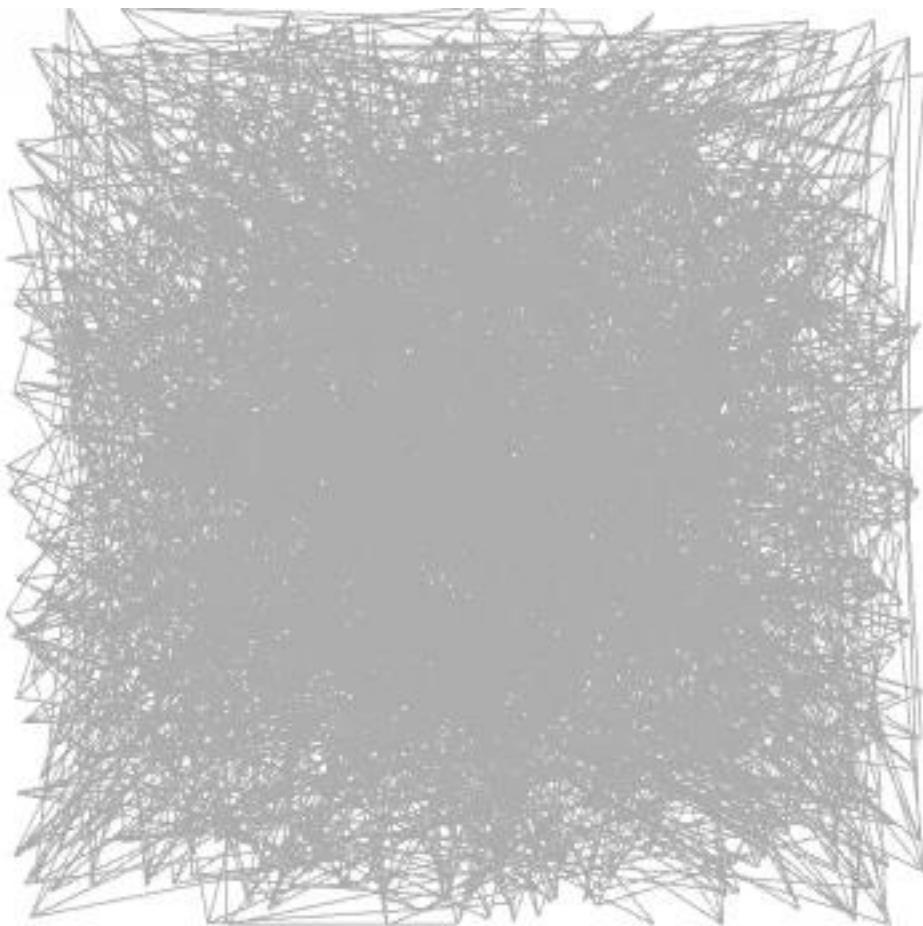
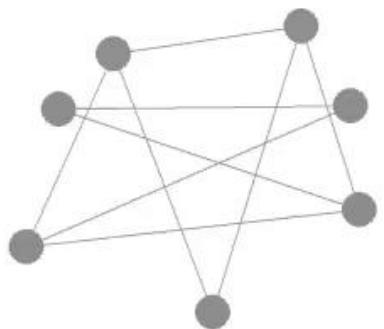


Forces Directed Methods

Force-directed graph drawing

- Use a physical analogy to draw graphs
 - View a graph as a system of bodies with forces acting between the bodies.
 - Vertices: particles that exert force on each other
 - Edges: forces between particles
- The algorithm seeks a position of each body with
 - minimal energy
 - sum of the forces on each body is zero





In general, force-directed methods have two parts: model & algorithm

1. Model:

- Force system
 - The model: a force system defined by vertices & edges
 - The algorithm: a technique for finding an equilibrium configuration, where the sum of the forces on each vertex is zero.

and/or

- Energy system
 - The model: maybe defined as an energy system
 - The algorithm: a technique for finding a configuration with locally minimal energy

力系统

- 模型: 由顶点和边界定义的力系统

- 算法: 一种寻找平衡配置的技术, 其中每个顶点上的力的总和为零。
和/或

• 能源系统

- 模型: 可能被定义为能源系统

- 算法: 用于查找具有局部最小能量的配置的技术

2. Algorithm:

- Many different optimization methods

Force directed methods:

1. Barycenter method
2. Spring & electrical force
3. An example using forces: metro maps
4. FADE (Barnes-Hutt)
5. Spectral methods

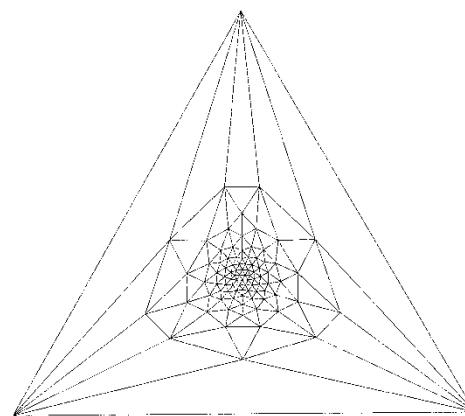
1. Barycenter Method [Tutte 1960 - 1963]

1. Model

- Use Hooke's law springs with natural length 0
- Pin down some vertices

2. Algorithm

- Solve linear equations



1. The Model

For a graph $G = (V, E)$ where each vertex u has a position $p_u = (x_u, y_u)$:

a) Use Hooke's law springs with natural length 0:

- The energy in an edge (u, v) (a spring) is
$$\text{energy}(u, v) = d(p_u, p_v)^2 = (x_u - x_v)^2 + (y_u - y_v)^2;$$
here $d(p_u, p_v)$ is the Euclidean distance between p_u and $p_v.$
- The total energy in the drawing is the sum of the energies in all the edges:

$$\text{Energy}(\mathbf{p}) = \sum_{(u,v) \in E} \text{energy}(u, v)$$

b) Pin down some vertices:

- A subset A of vertices if fixed.
- That is, for a vertices $a \in A \subset V$, p_a is chosen *a priori*.
- That is, for $a \in A$, p_a is part of the input.

2. Algorithm

- Minimize energy by solving linear equations →

Barycenter Method

How to minimize energy:

- Choose a location $\mathbf{p}(\mathbf{u}) = (x_{\mathbf{u}}, y_{\mathbf{u}})$ for each $\mathbf{u} \in V - A$ to minimize

$$Energy(\mathbf{p}) = \sum_{(\mathbf{u}, \mathbf{v}) \in E} d(\mathbf{u}, \mathbf{v})^2 = \sum_{(\mathbf{u}, \mathbf{v}) \in E} (x_{\mathbf{u}} - x_{\mathbf{v}})^2 + (y_{\mathbf{u}} - y_{\mathbf{v}})^2$$

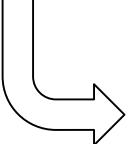
- Note that the minimum is unique, and occurs when

$$\frac{\partial (energy(\mathbf{p}))}{\partial x_{\mathbf{u}}} = \mathbf{0} \quad \text{and} \quad \frac{\partial (energy(\mathbf{p}))}{\partial y_{\mathbf{u}}} = \mathbf{0}$$

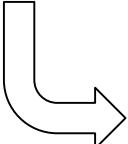
for each $\mathbf{u} \in V - A$.

For each vertex $u \in V - A$, for the x -coordinate x_u , minimum energy occurs when:

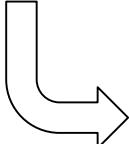
$$\frac{\partial (\text{Energy}(p))}{\partial x_u} = 0$$



$$\frac{\partial}{\partial x_u} \left(\sum_{(u,v) \in E} (x_u - x_v)^2 + (y_u - y_v)^2 \right) = 0$$

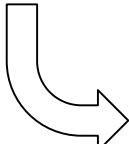


$$\sum_{v \in N_u} 2(x_u - x_v) = 0$$



$$\sum_{v \in N_u} x_u - \sum_{v \in N_u} x_v = 0$$

Barycentre
equations



$$deg(u)x_u - \sum_{v \in N_u} x_v = 0$$

Barycenter Method

For each vertex $u \in V - A$, we have an equations in the x direction:

$$\deg(v)x_v - \sum_{u \in N(v)} x_u = 0$$

(and a similar equation in the y direction)

That is:

$$x_v = \frac{1}{\deg(v)} \sum_{u \in N(v)} x_u$$

barycentre equations

Taking both the x direction and the y direction, the barycentre equations say that

“the position p_v of v is barycentre of the positions p_u of the neighbors u of v . ”

“Pin down” some vertices:- the vertex set V is partitioned into two sets:

a) fixed vertex set A

- at least 3 vertices, ie, $|A| \geq 3$.
- nailed down, pinned down, fixed, unmovable
- The fixed vertices and their positions may be chosen in different ways; for the moment just assume that they are in general position.

b) free vertex set $V - A$

Barycenter Method

To find a position for each vertex $u \in V - A$,

we must solve two sets of $|V - A|$ linear equations in $2|V - A|$ unknowns:

For all free vertices v :

$$\begin{cases} x_v = \frac{1}{\deg(v)} \sum_{u \in N(v)} x_u \\ y_v = \frac{1}{\deg(v)} \sum_{u \in N(v)} y_u \end{cases} \quad \leftarrow \text{\underline{barycentre equations}}$$

We can write the barycentre equations in matrix form.

Background: graph matrices

- » Let M be the adjacency matrix of $G = (V, E)$; that is,

$$M_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- » Let D be the degree matrix of G ; that is, D is the diagonal matrix of degrees, that is,

$$D_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- » Then the Laplacian L of G is $L = D - M$, that is,

$$L_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } (i, j) \in E \\ 0 & \text{if } i \neq j \text{ and } (i, j) \notin E \end{cases}$$

The barycentre equations (for the x direction):

$$\deg(v)x_v - \sum_{u \in N(v) \cap (V-A)} x_u = c_v.$$

In matrix form this is:

barycentre equations

$$L'x = c_x$$

where

- » L' is a matrix formed from the Laplacian by deleting rows and columns corresponding to fixed vertices.
- » x is the column vector indexed by the vertices of $V - A$.
- » c_x is a constant column vector.

And we have similar equations for the y direction.

Barycenter Method

1. Model:

- Fix a set A of vertices
- Place every other vertex at the barycentre of its neighbours; that is,
choose the position (x_i, y_i) of every free vertex $v_i \in V - A$ to satisfy the barycentre equations

$$L'x = c_X,$$

$$L'y = c_Y.$$

2. Algorithm:

- Solve the barycentre equations, that is, compute L'^{-1} .

The matrix L' :

- Submatrix or the Laplacian matrix of the input graph
- Using Kirchhoff matrix-tree theorem, the inverse exists if and only if for every vertex in $V - A$ there is a path to a vertex in A .
- Weakly diagonally dominant: for every i ,

$$|D_{ii}| \geq \sum_{i \neq j} |D_{ij}|$$

with

$$|D_{ii}| > \sum_{i \neq j} |D_{ij}|$$

for at least 3 values of i .

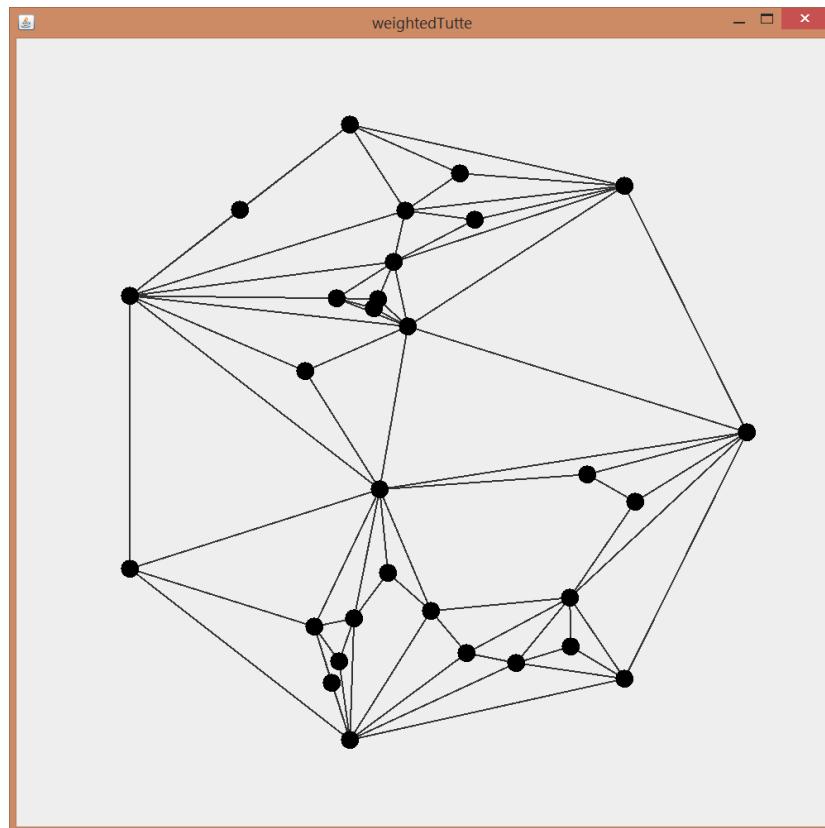
How to solve the barycentre equations:

- Gaussian elimination: $O(n^3)$
- Fast matrix multiplication methods: $\sim O(n^{2.5})$ or less
- For sparse Laplacian matrices (ie, graphs with a linear number of edges, such as planar graphs): Nested dissection methods $O(n^{1.5})$.
- Speilman-Teng methods: perhaps $O(n \log n)$.

- Gauss-Seidel / Jacobi methods (iterative approximations)
 - » In practice, the Gauss-Seidel / Jacobi methods are very simple and work well because of properties of Laplacian matrices.

Choosing and placing the fixed set A :

- If G is a planar embedding then we choose A to be an outside face
- Normally we place A at the vertices of a convex polygon



Theorems

Theorem [Tutte 1963] Suppose that G is a triconnected planar graph, f is a face in a planar embedding of G , and P is a strictly convex planar drawing of f . Then applying the barycentre method with the vertices of f fixed and positioned according to P , yields a planar drawing of G in which every face is convex.

Theorem (Informally stated): the barycentre algorithm applied to a graph G displays every dihedral automorphism of G that fixes A , as a symmetry of the drawing.

Barycentre method

Advantages

- Elegant and easy to implement
- In practice, the simple Gauss-Seidel / Jacobi iterations converges quickly
- One can use weights on the edges to implement many things:
 - Logical fish-eye lens
 - Constrain chosen paths to be straight lines
 - Cluster nodes together
 - Smooth animation for in-betweening

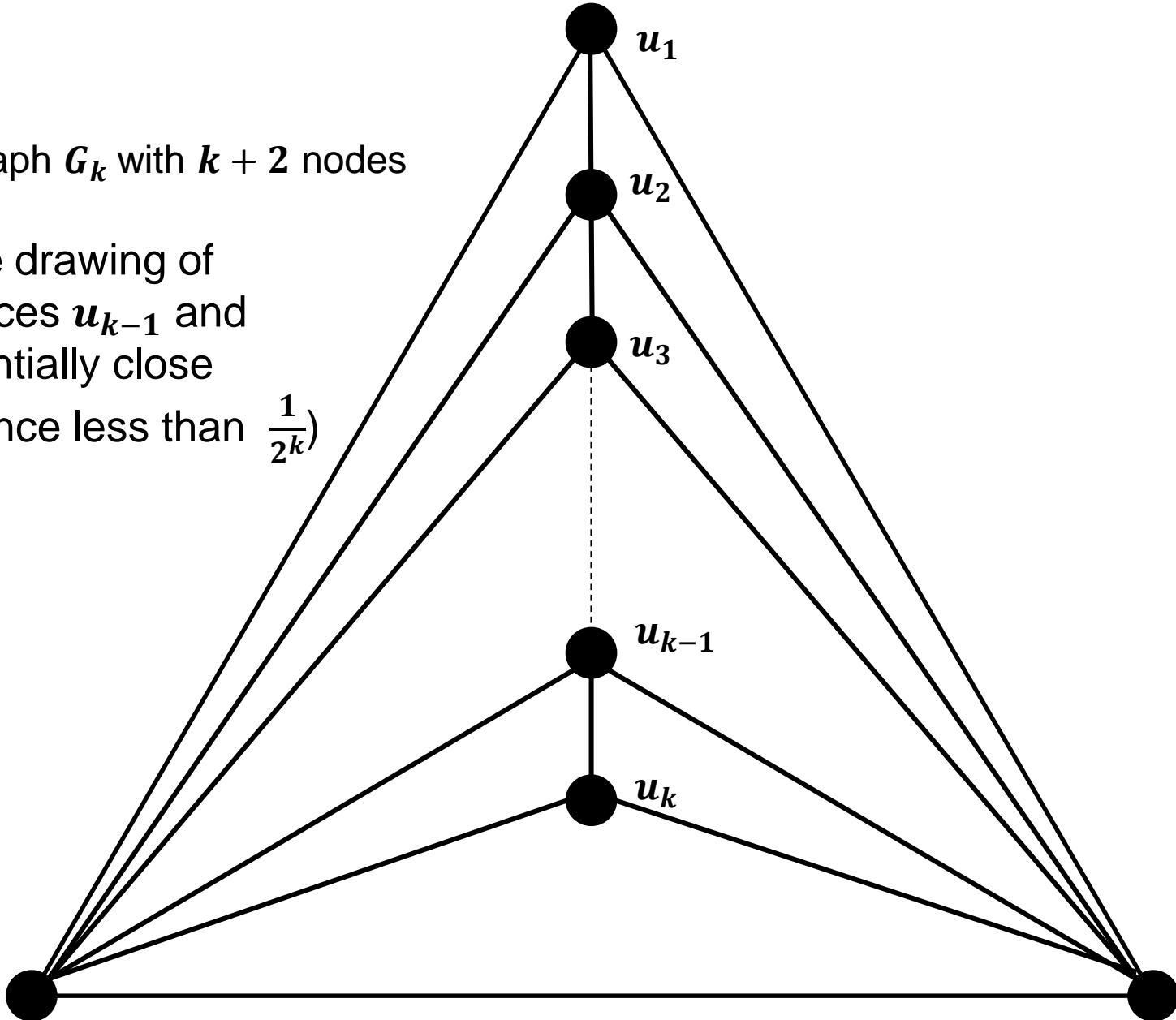
Disadvantages

- Poor vertex resolution

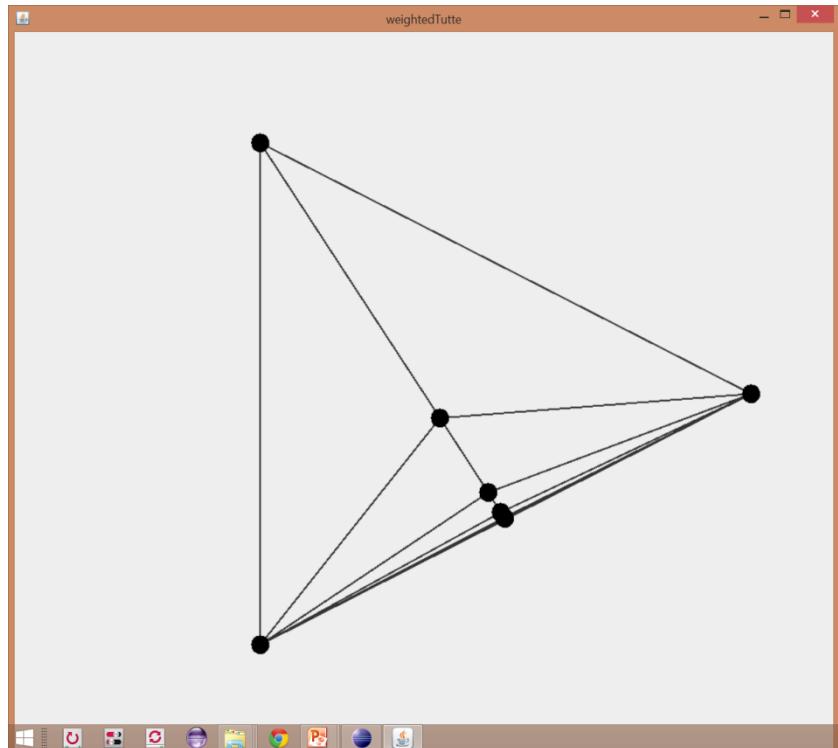
Barycenter Method

Consider this graph G_k with $k + 2$ nodes

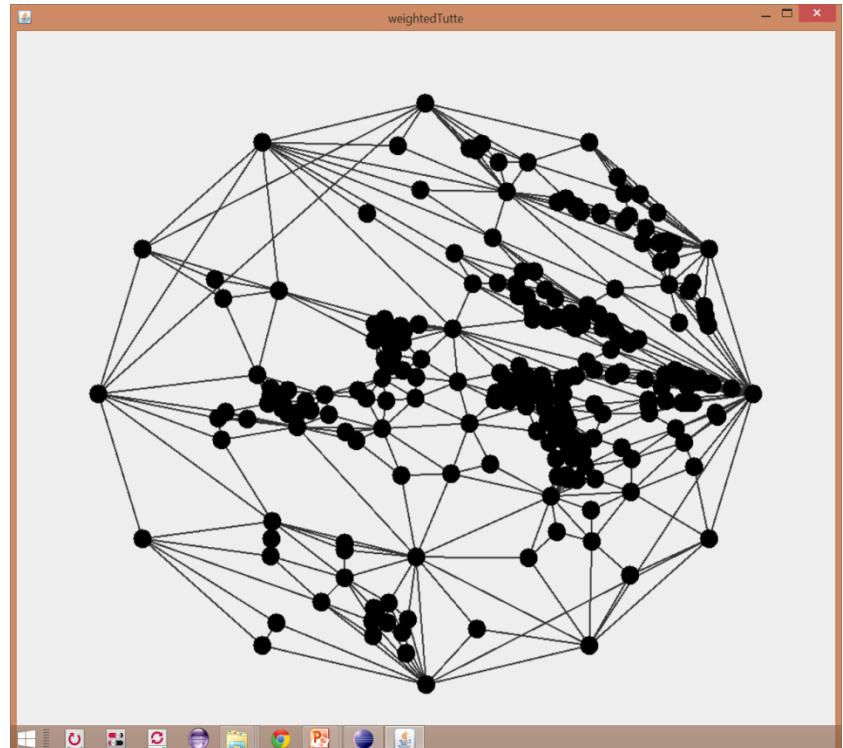
In a barycentre drawing of graph G_k , vertices u_{k-1} and u_k are exponentially close together (distance less than $\frac{1}{2^k}$)



$k = 6$



About 300 vertices



2. Springs & Electrical Forces

Springs and electrical forces

1. Model:

- Edges are springs
- Vertices are electrically charged particles that repel each other

2. Algorithm:

- Many different optimization methods
- The simplest algorithm is “follow-your-nose”

1. Model

Model edges and vertices with Hooke's Law spring forces plus electrical forces

- Vertex: modelled as an electrically charged particle.
- Edge: modelled as a spring

Force on a particle v :-

$$\mathbf{F}(v) = \sum_{u \in N(v)} \mathbf{f}_{uv} + \sum_{u \in V, u \neq v} \mathbf{g}_{uv}$$

where

$N(v)$ is the set of neighbours of v ;

\mathbf{f}_{uv} is the force exerted on v by the (Hooke's law) spring between u and v corresponding to the edge (u, v) ;

\mathbf{g}_{uv} is the force exerted on v by the electrical repulsion between u and v .

Notes

- The spring force ensure that edges have about the right length.
- The electrical force ensures that vertices are not too close to each other.
- $\mathbf{F}(v) = (\mathbf{F}_x(v), \mathbf{F}_y(v))$ is a vector.

The force on a particle v is: -

$$F(v) = \sum_{u \in N(v)} \alpha_{uv} (d(p_u, p_v) - \beta_{uv}) i_{uv} + \sum_{u \in V, u \neq v} \frac{\gamma_{uv}}{(d(p_u, p_v))^2} i_{uv}$$

where

p_u is the position of vertex u ,

p_v is the position of vertex v ,

$d(p_u, p_v)$ is the Euclidean distance between p_u and p_v ,

i_{uv} is the unit vector in the direction from p_u to p_v .

and

α_{uv} , β_{uv} and γ_{uv} are constants.

That is, the force on a particle v is: -

➤ in the x direction $F_x(v)$ is

$$\sum_{u \in N(v)} \alpha_{uv} (d(p_u, p_v) - \beta_{uv}) \left(\frac{x_u - x_v}{d(p_u, p_v)} \right) + \sum_{u \in V, u \neq v} \frac{\gamma_{uv}}{(d(p_u, p_v))^2} \left(\frac{x_u - x_v}{d(p_u, p_v)} \right)$$

➤ in the y direction $F_y(v)$ is

$$\sum_{u \in N(v)} \alpha_{uv} (d(p_u, p_v) - \beta_{uv}) \left(\frac{y_u - y_v}{d(p_u, p_v)} \right) + \sum_{u \in V, u \neq v} \frac{\gamma_{uv}}{(d(p_u, p_v))^2} \left(\frac{y_u - y_v}{d(p_u, p_v)} \right)$$

This gives $2|V|$ equations.

- Note: The unit vector in the direction from u to v is $\left(\left(\frac{x_u - x_v}{d(p_u, p_v)} \right), \left(\frac{y_u - y_v}{d(p_u, p_v)} \right) \right)$.

$$\mathbf{F}(v) = \sum_{u \in N(v)} \alpha_{uv} (d(p_u, p_v) - \beta_{uv}) \mathbf{i}_{uv} + \sum_{u \in V, u \neq v} \frac{\gamma_{uv}}{(d(p_u, p_v))^2} \mathbf{i}_{uv}$$

Note:

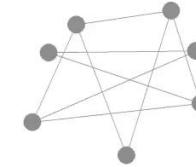
- α_{uv} , β_{uv} and γ_{uv} have physical interpretations:
 - α_{uv} is the *stiffness* of the spring corresponding to the edge (u, v) .
 - β_{uv} is the *natural length* of the spring corresponding to the edge (u, v) .
 - γ_{uv} is the *strength* of the electrical repulsion between u and v .
- We can map attributes of the vertices and edges to α_{uv} , β_{uv} and γ_{uv} ; for example:
 - β_{uv} is the ideal length of the edge (u, v) , as defined by the application domain.
 - A high value of α_{uv} means that the edge (u, v) between u and v is important; a low value of α_{uv} means that this edge is unimportant.

2. Algorithm

We need to compute a position \mathbf{p}_v such that $\mathbf{F}(\mathbf{v}) = \mathbf{0}$ for each $v \in V$, where:

$$\mathbf{F}(v) = \sum_{u \in N(v)} \alpha_{uv} (\mathbf{d}(\mathbf{p}_u, \mathbf{p}_v) - \beta_{uv}) \mathbf{i}_{uv} + \sum_{u \in V, u \neq v} \frac{\gamma_{uv}}{(\mathbf{d}(\mathbf{p}_u, \mathbf{p}_v))^2} \mathbf{i}_{uv}$$

- This is an energy minimization problem
- The equations are not linear.
- There are many techniques to find an equilibrium configuration (zero force on each node, or locally minimum energy).
- The solutions are not unique
 - That is, there may be locally minimum energy that is not globally minimum.



An algorithm example:

Simple algorithm (“Follow your nose”):

1. p_u = some initial position for each node u ;
2. Repeat
 - 2.1 $F_u := \mathbf{0}$ for each node u ;
 - 2.2 For each pair u, v of nodes
 - 2.2.1 calculate the force f_{uv} between u and v ;
 - 2.2.2 $F_u += f_{uv}$;
 - 2.2.3 $F_v += f_{uv}$;
 - 2.3 For each node u , $p_u += \epsilon F_u$;
- Until p_u converges for all u ;

- Not fast, but allows smooth animation.

Runtime (using the simple follow-your-node algorithm):

- Time to compute the zero-force configuration:-
 - The dominant time is calculating the electrical repulsion forces between all pairs of vertices: $O(|V|^2)$
 - Total time for an iteration: $O(|V|^2)$
- This is repeated for each iteration, until convergence or other conditions are met.
- In practice, a few hundred vertices means very slow runtime.
- More complex algorithms are available.

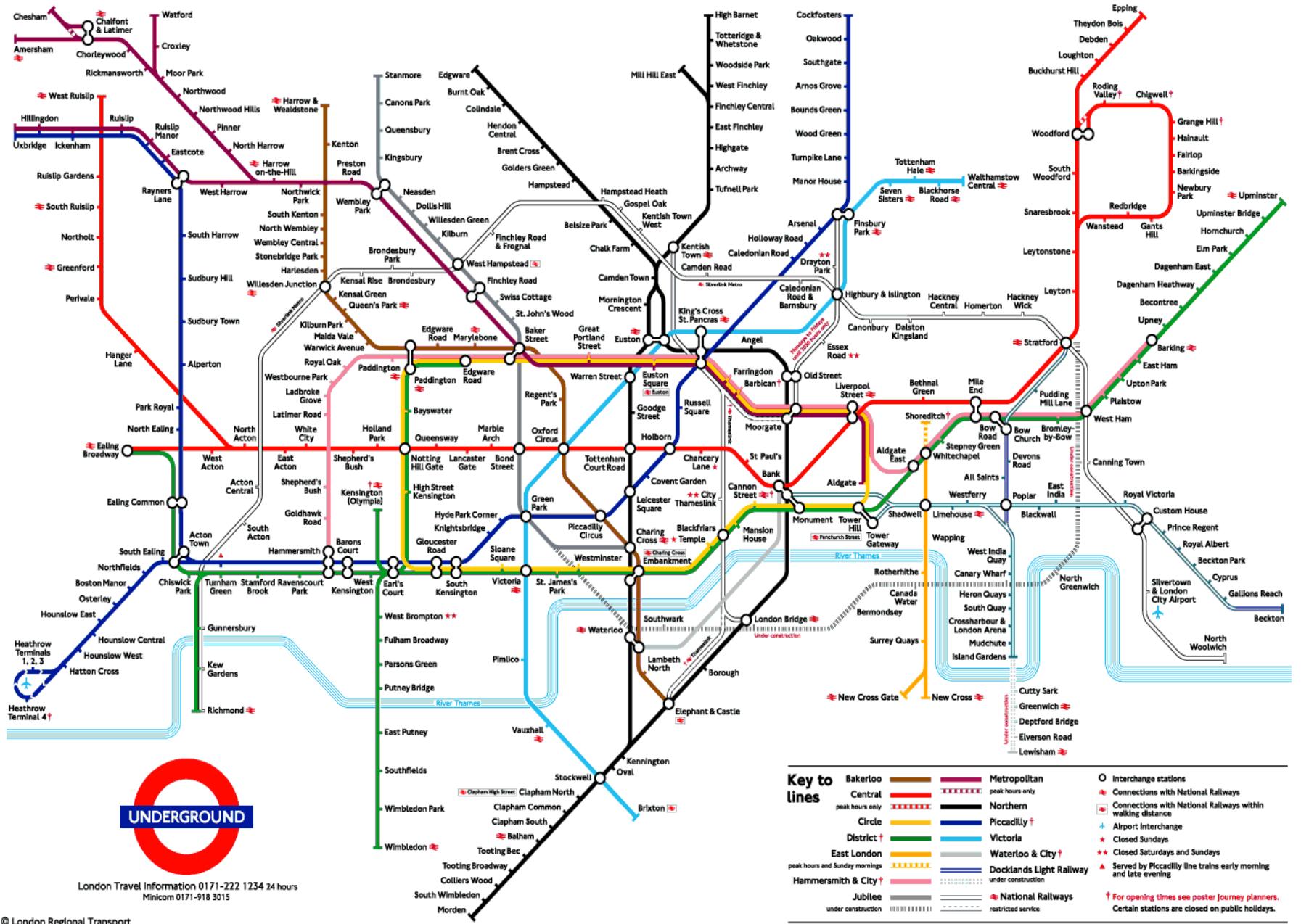
3. Example: using forces

The Metro Map Problem

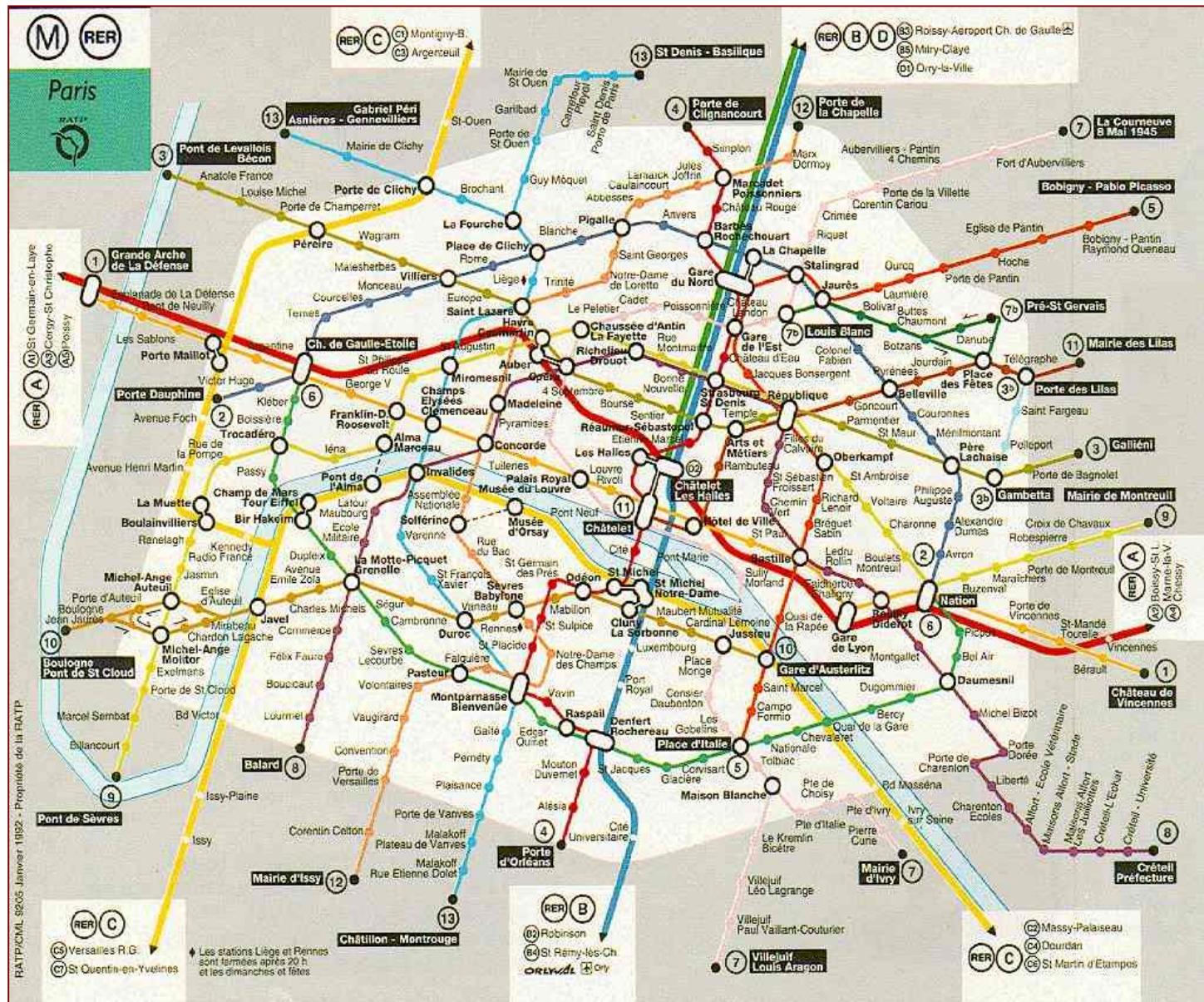
- Existing metro maps, produced manually by professional graphic artists, are **excellent examples** of network visualization
- *Can we produce good metro maps automatically?*

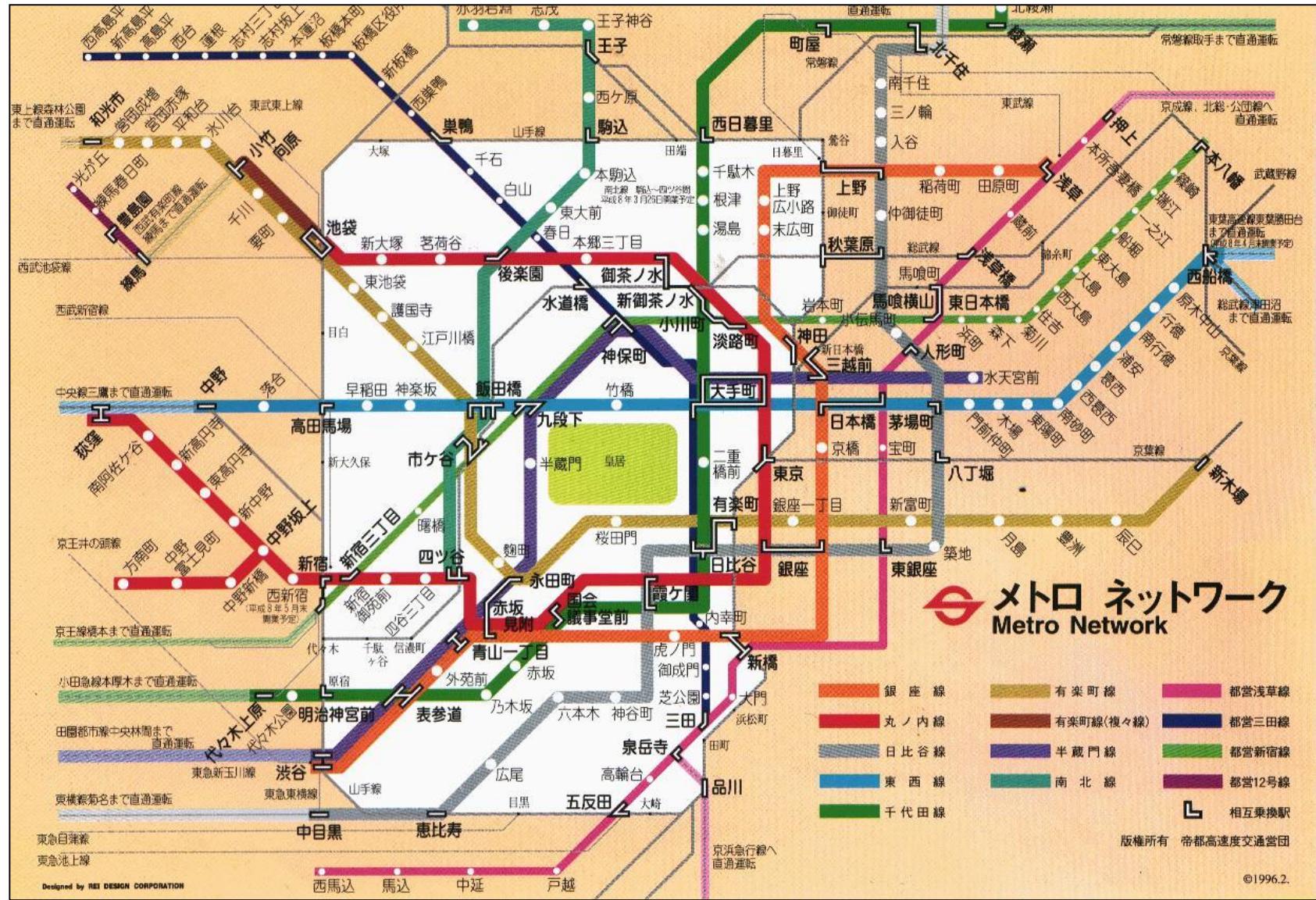


H. Beck, 1931

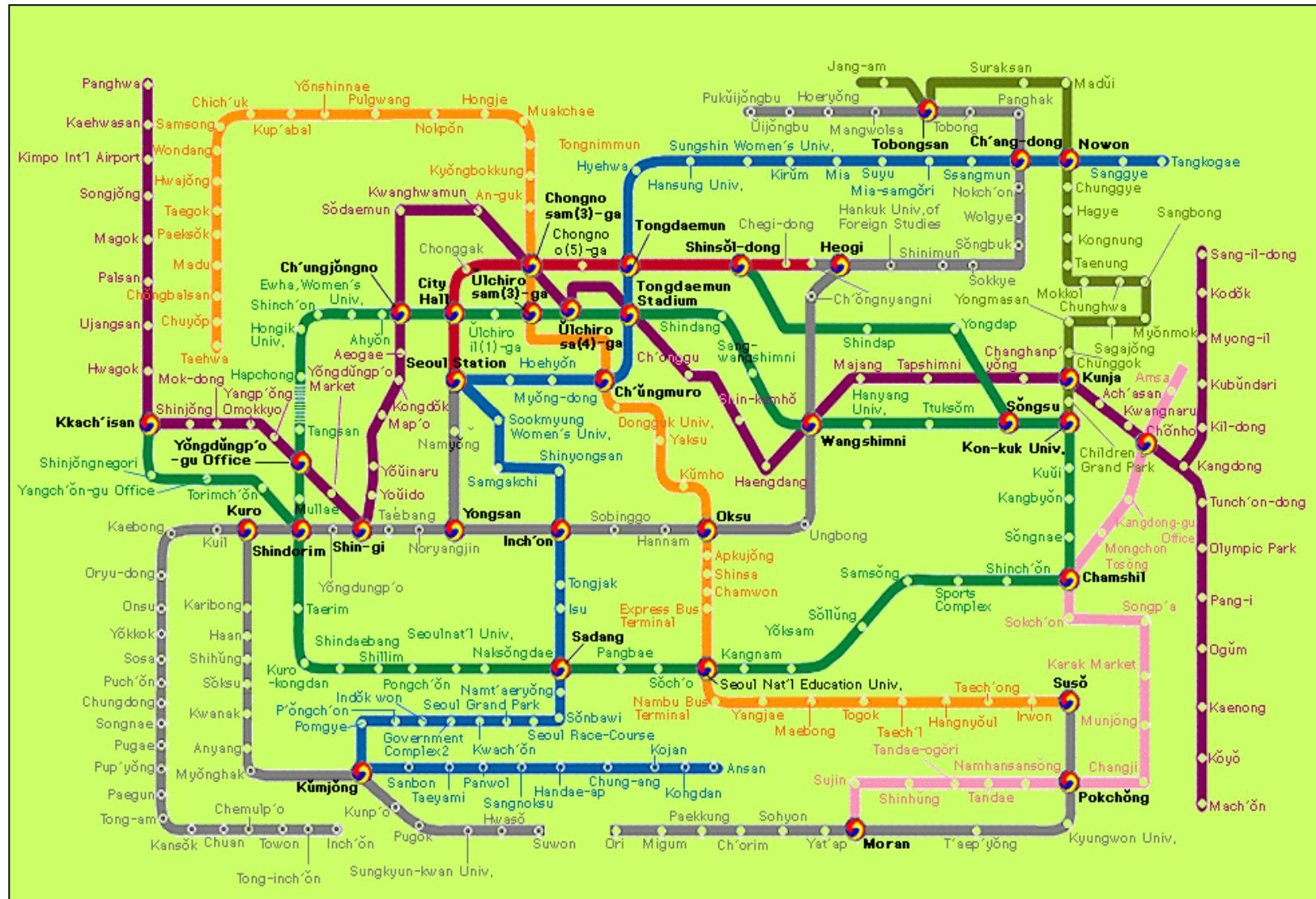




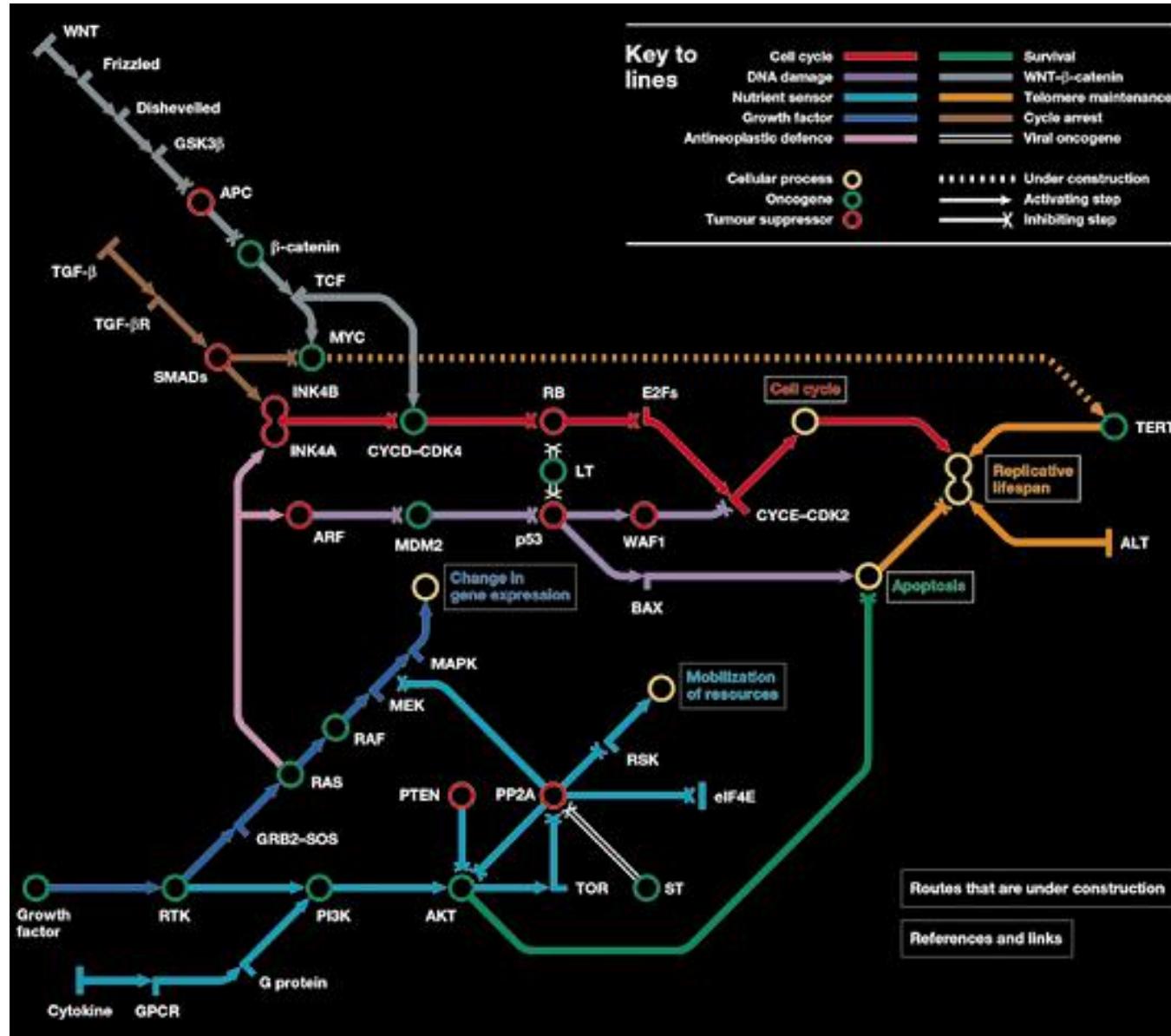




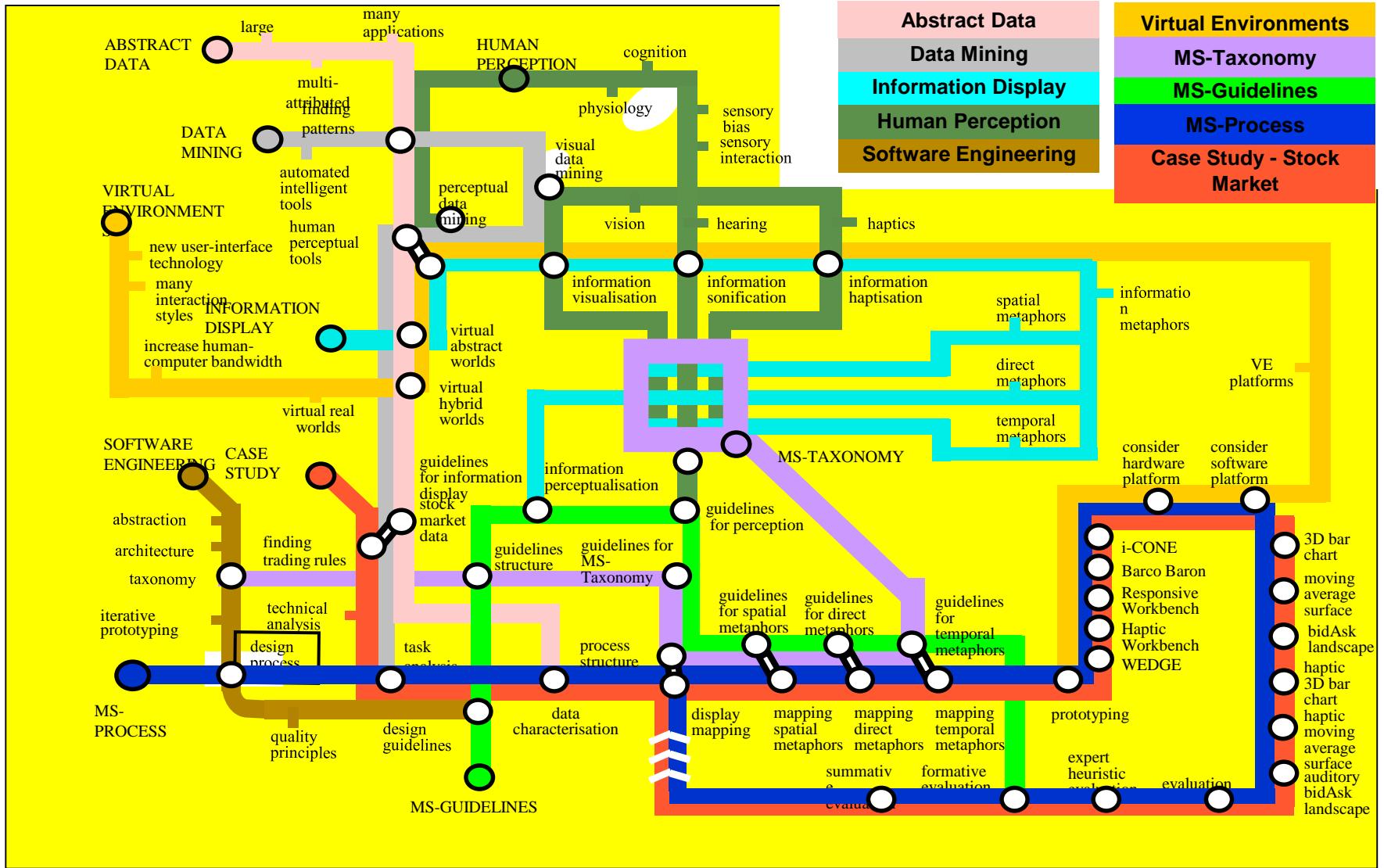




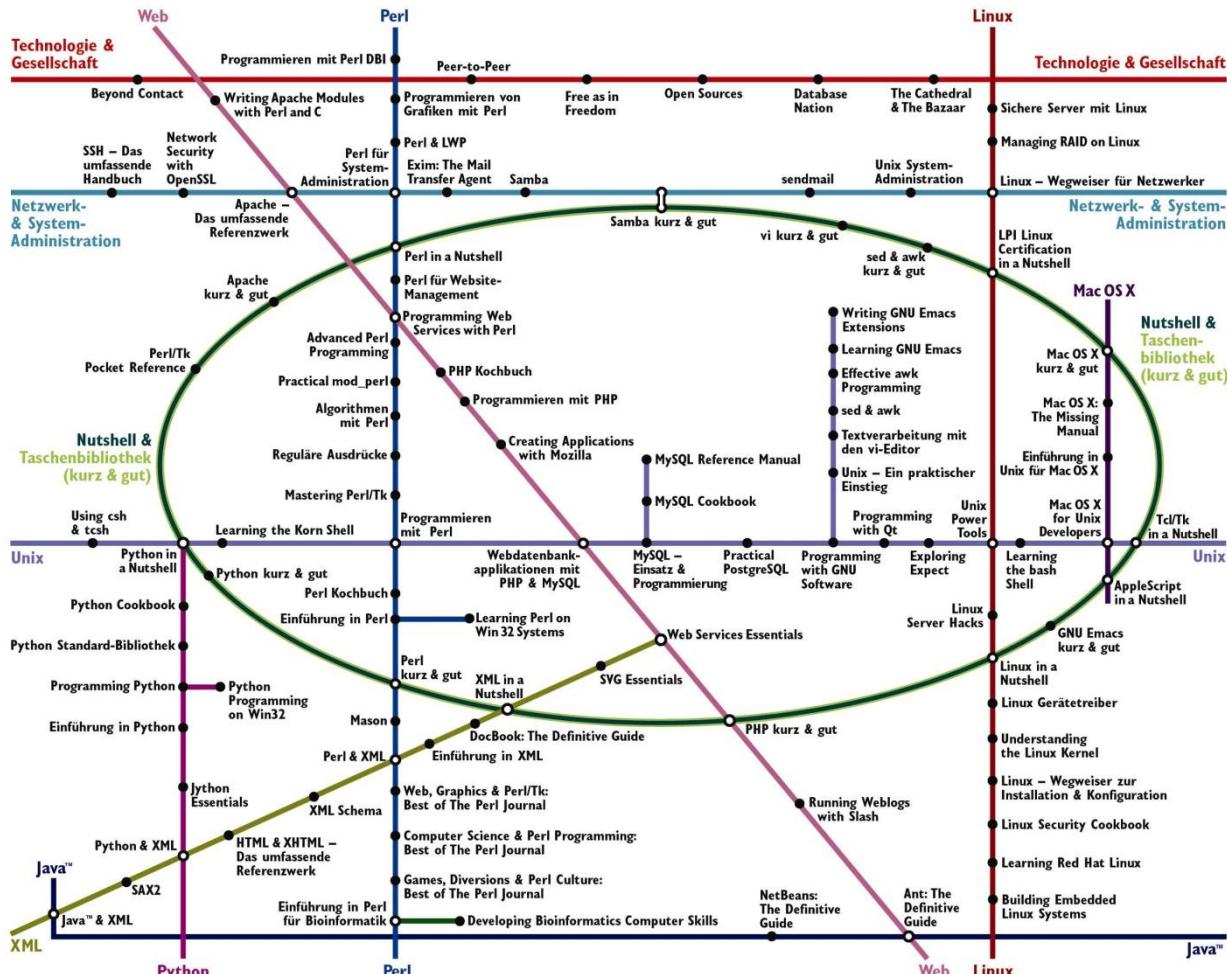
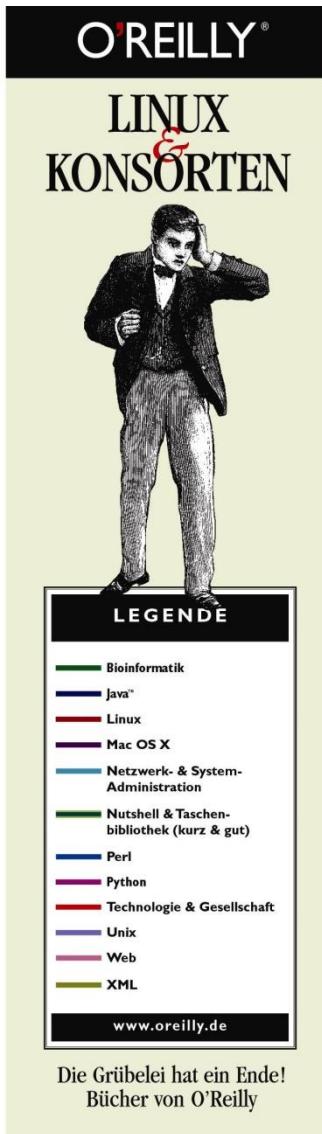




J. Hallinan

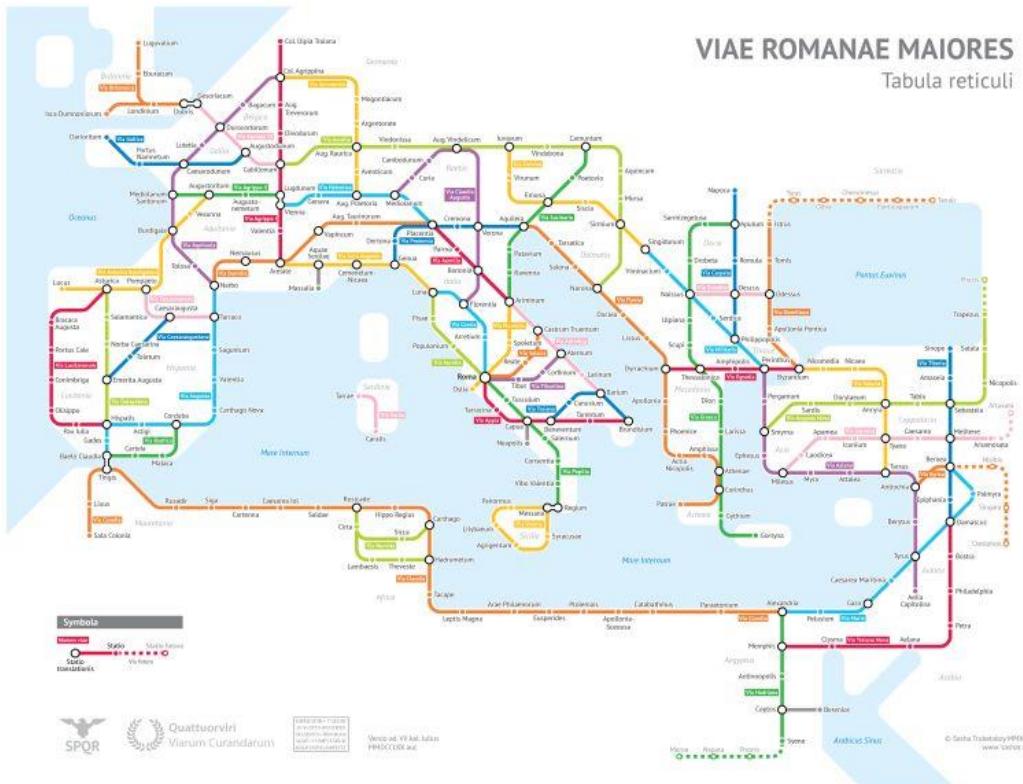


Keith Nesbitt



VIAE ROMANAEE MAIORES

Tabula reticuli



Scientific Question: Is there an E^3 computer algorithm that can produce a layout of a metro map network?

($E^3 = \text{Effective, Efficient, Elegant}$)

Metro map layout

Model

- ◆ Metro stations → steel rings
- ◆ Interconnections → magnetized springs
- ◆ Vertical/horizontal/45° magnetic field

Algorithm

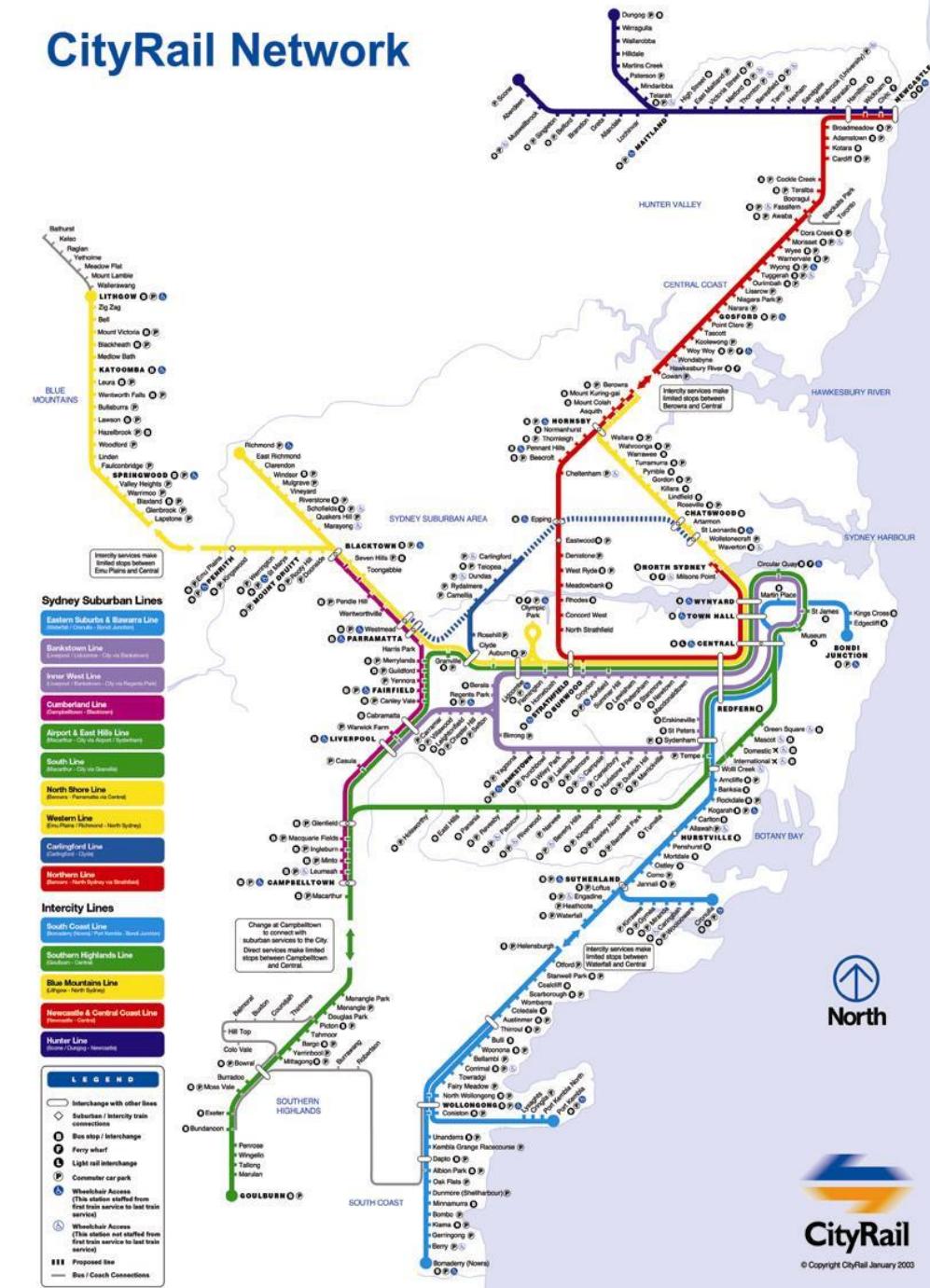
- ◆ Simple follow-your-nose
- ◆ Plus automatic label placement

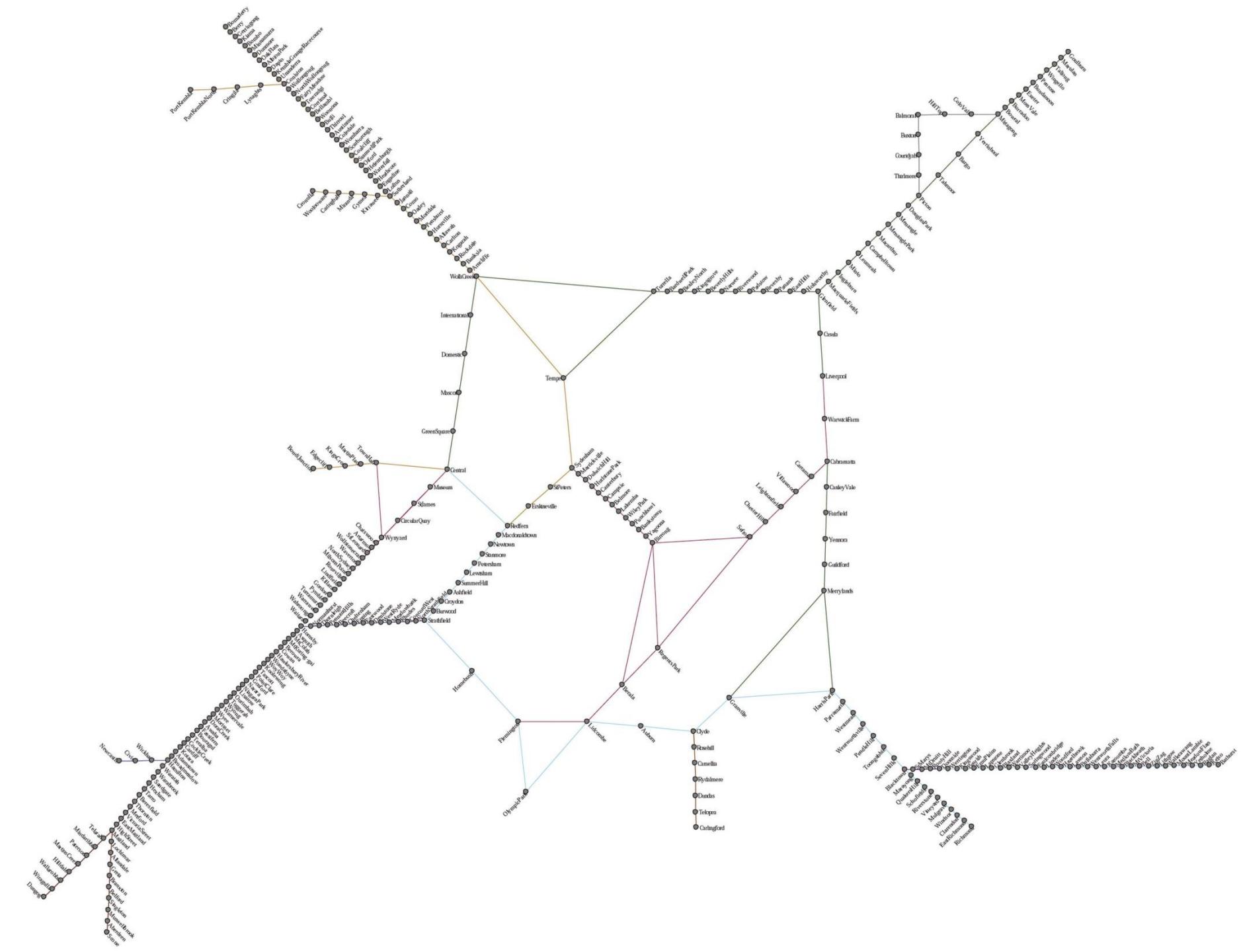
Elegance:

- This spring method is elegant

Effectiveness

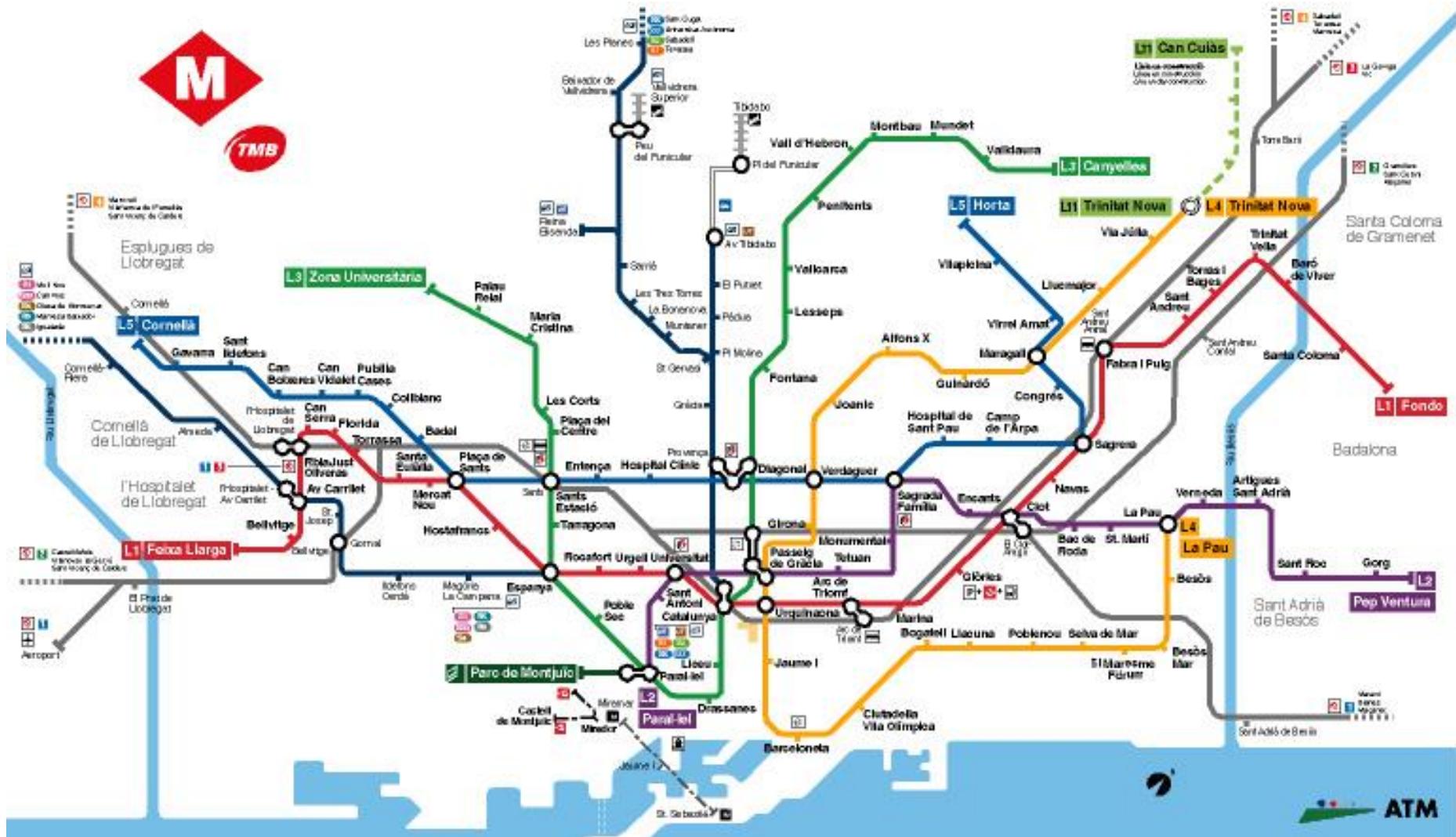
CityRail Network

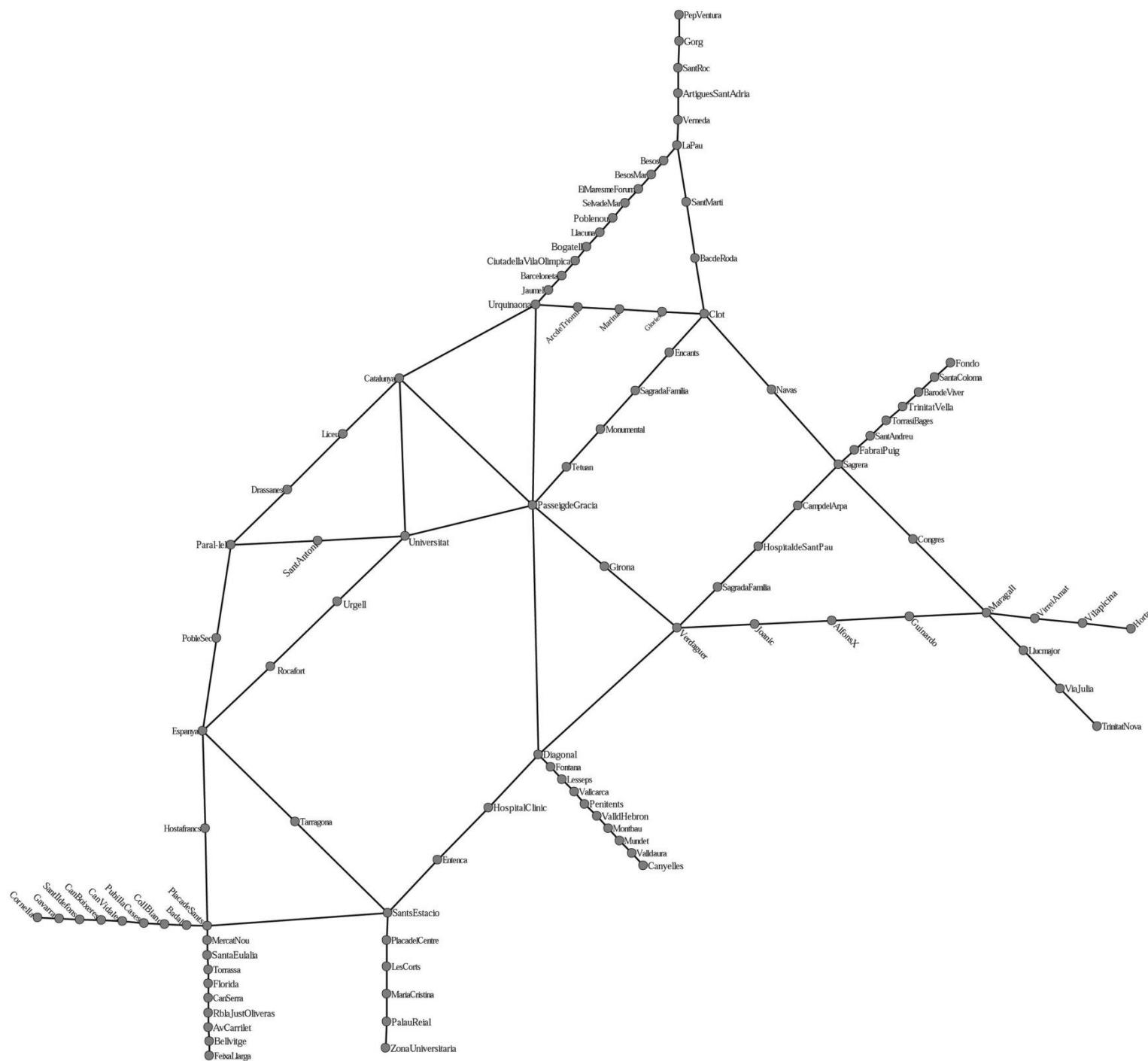


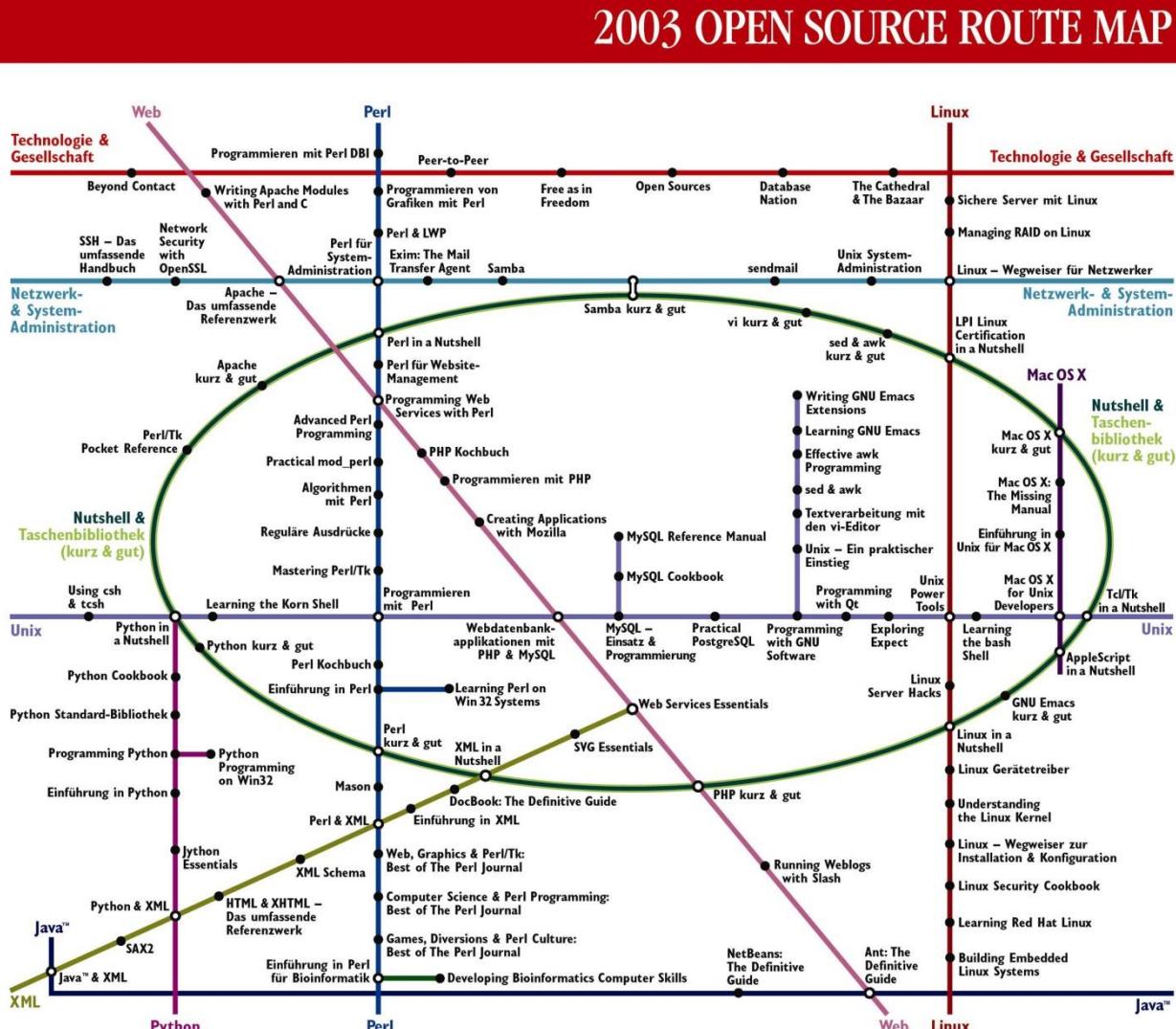
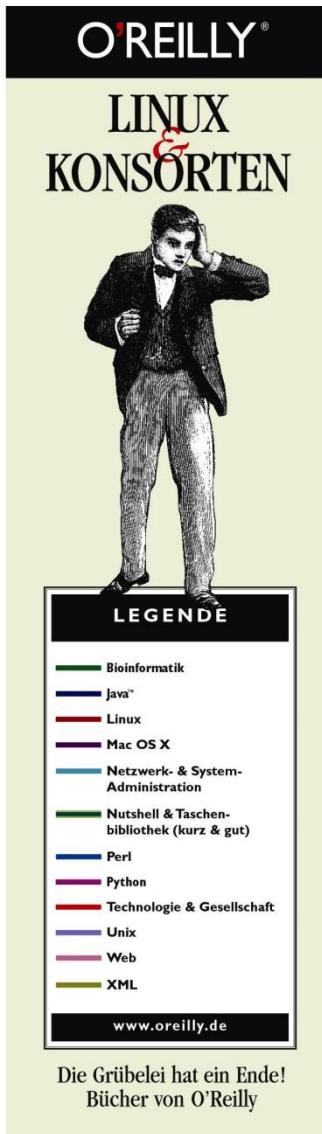


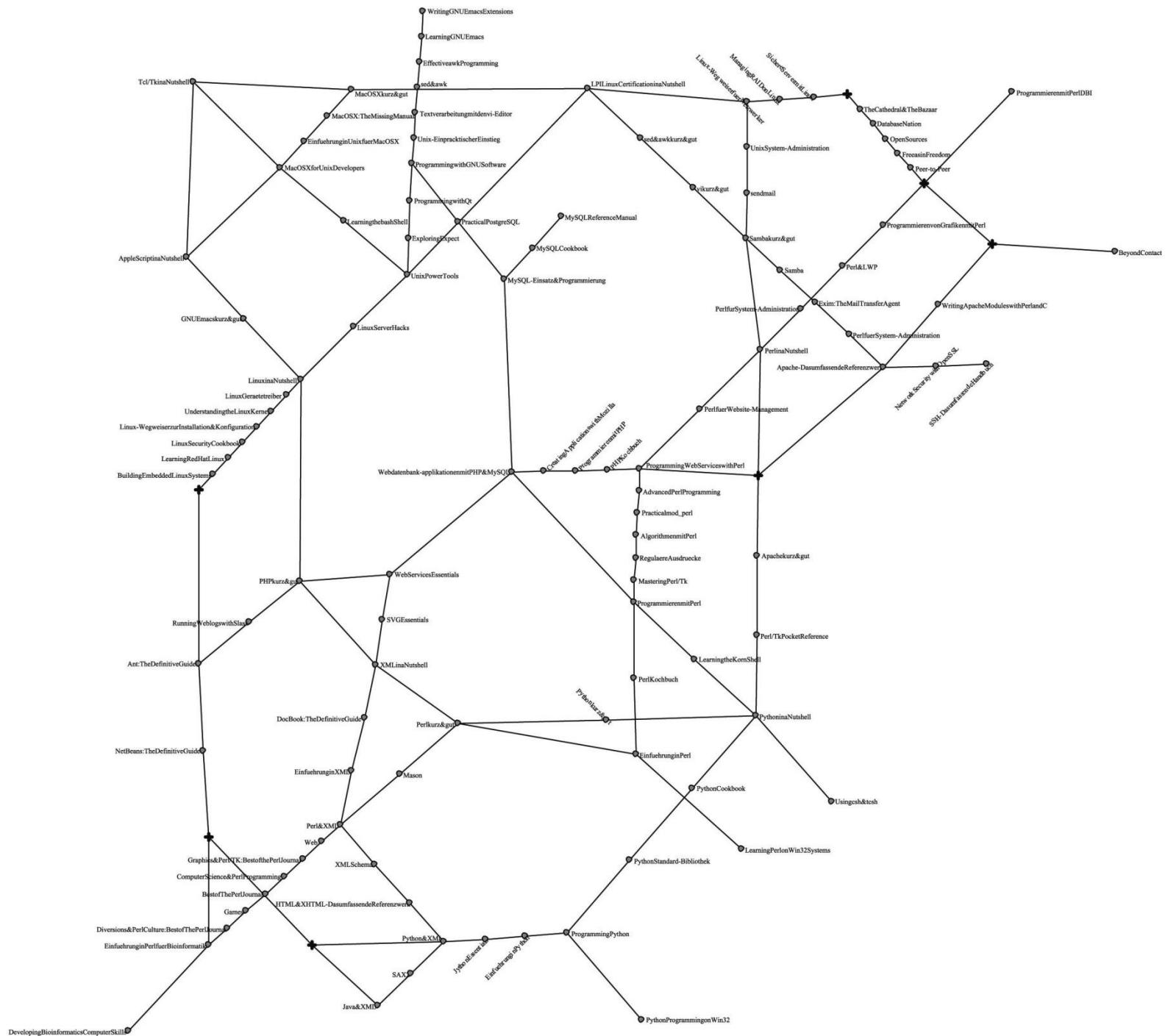


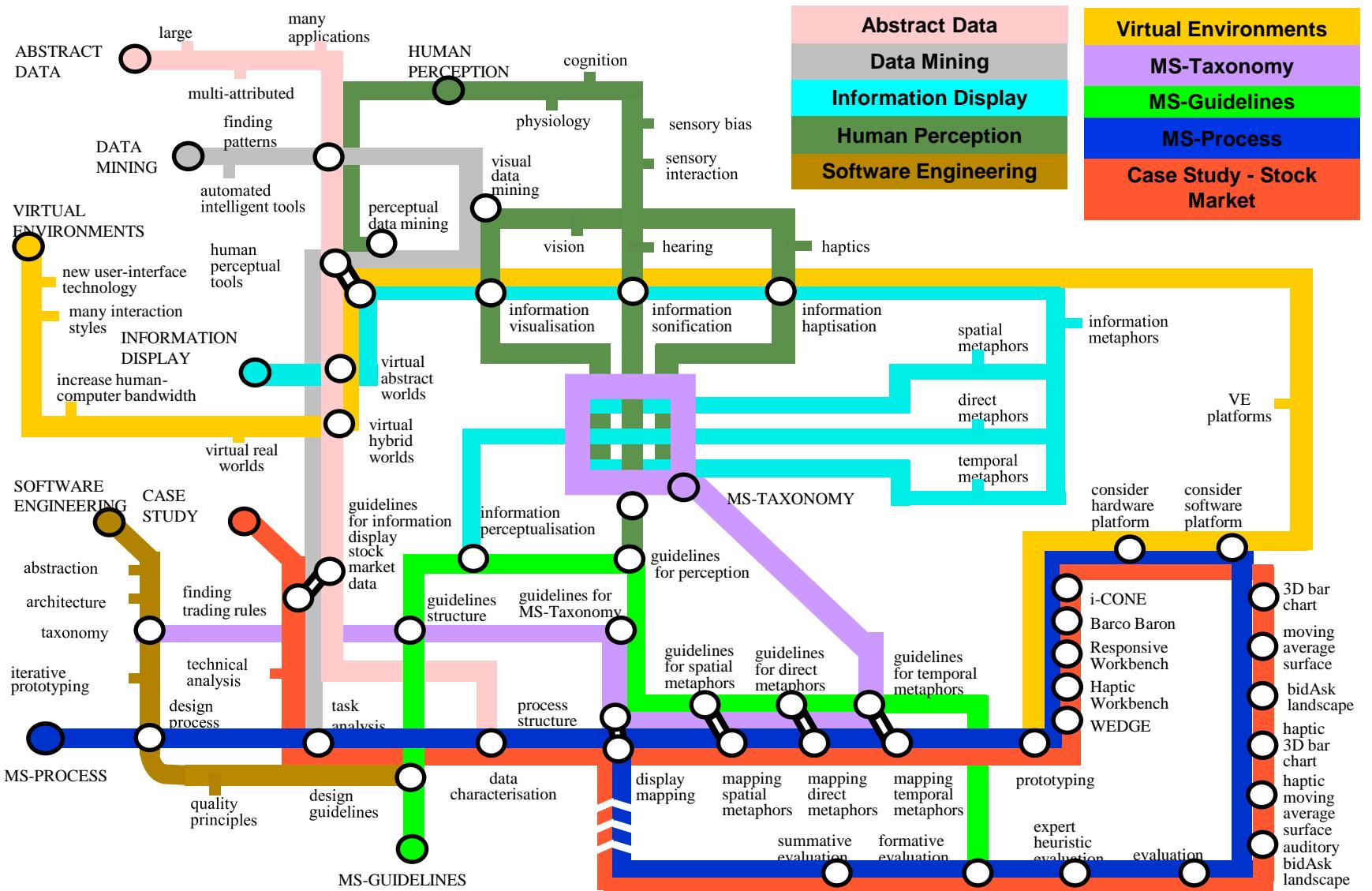
TMB

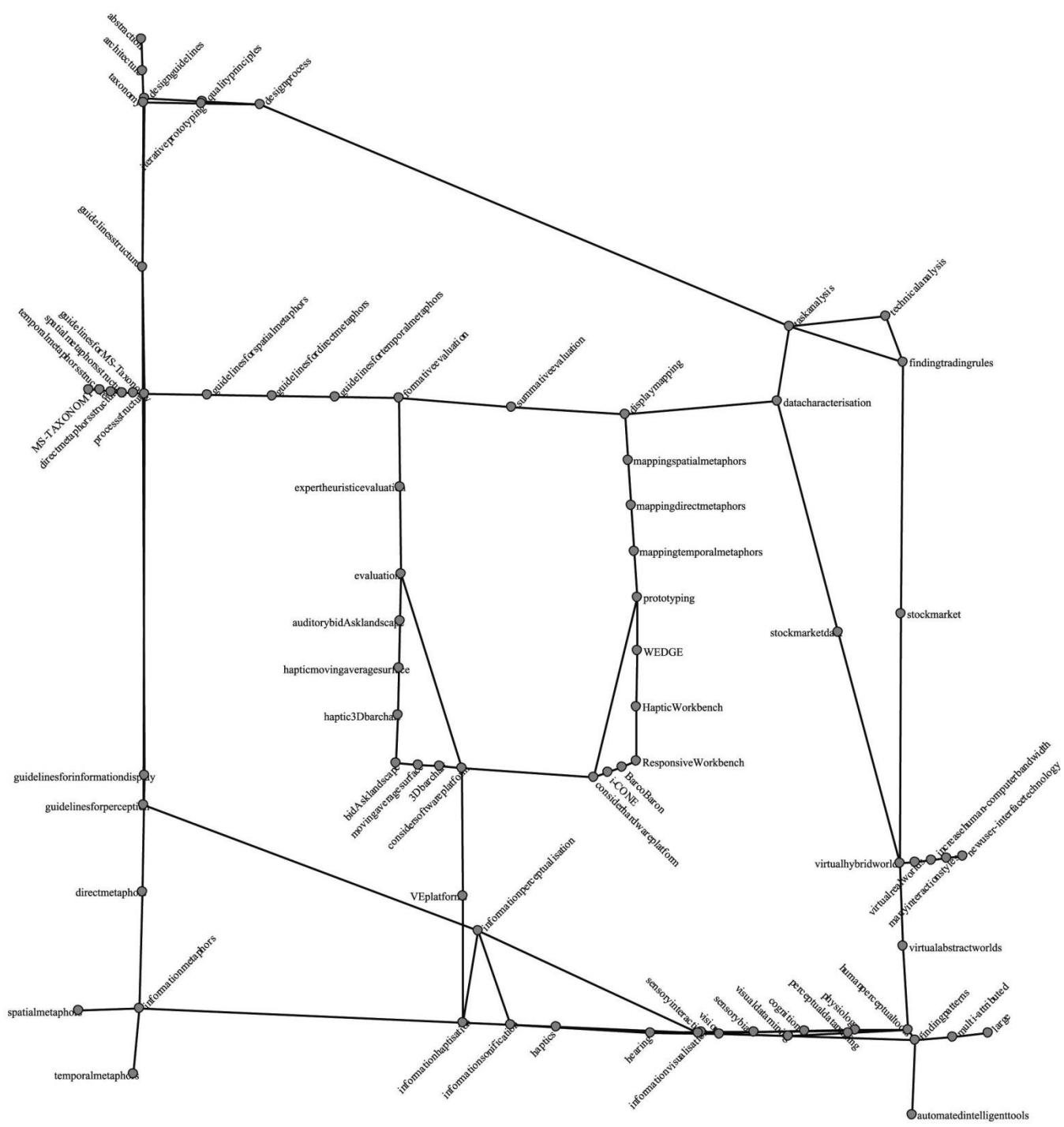


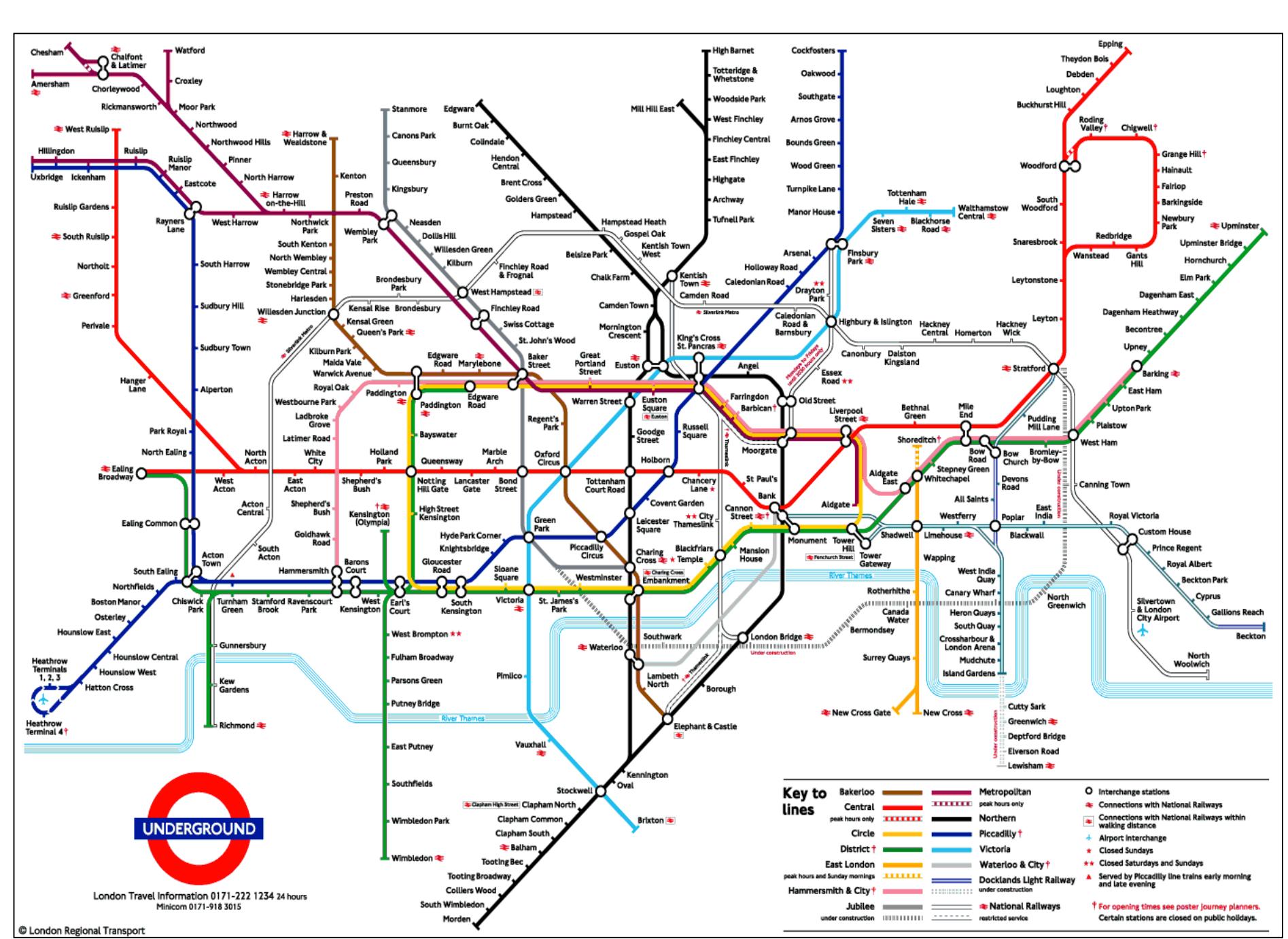


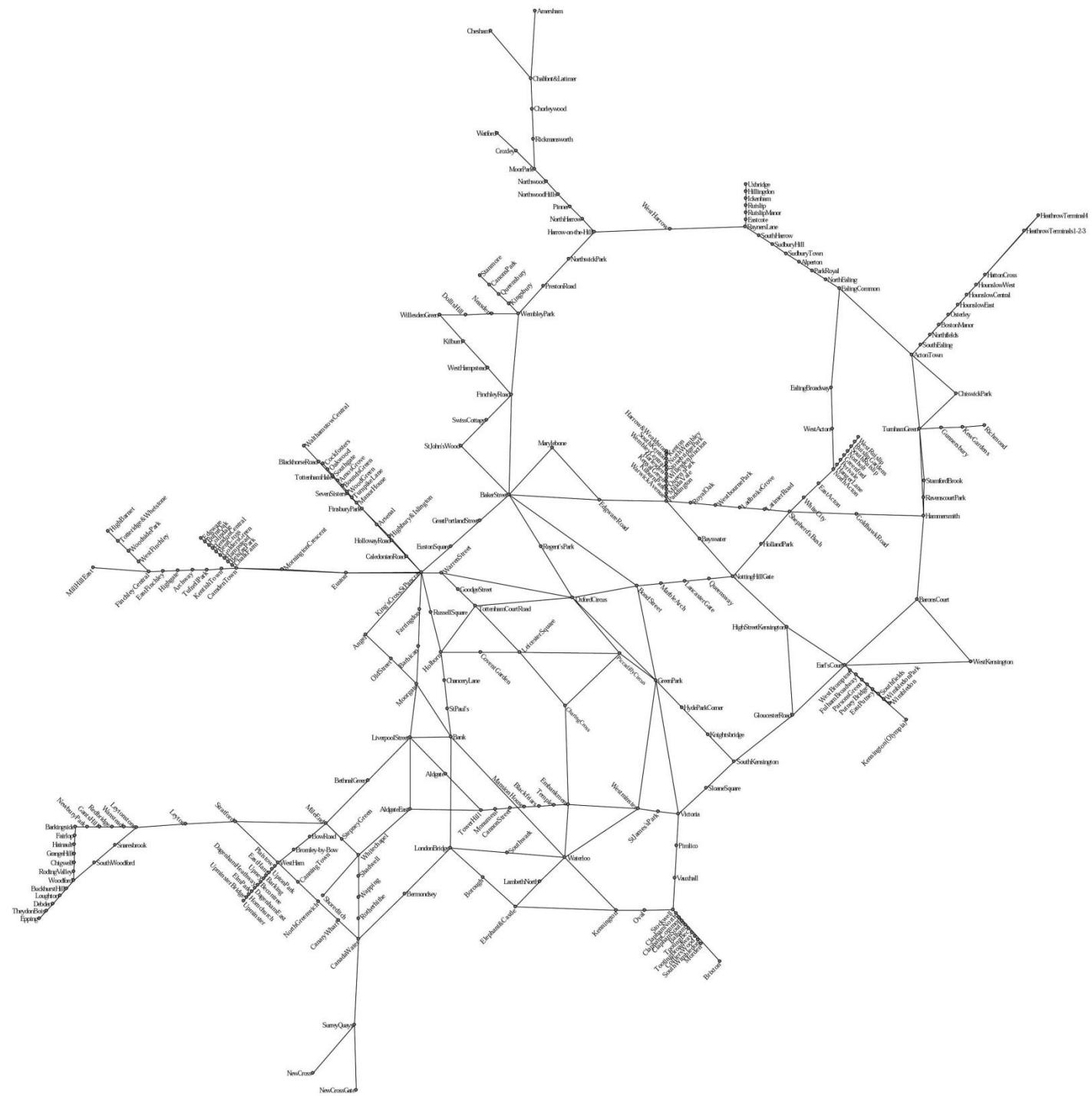










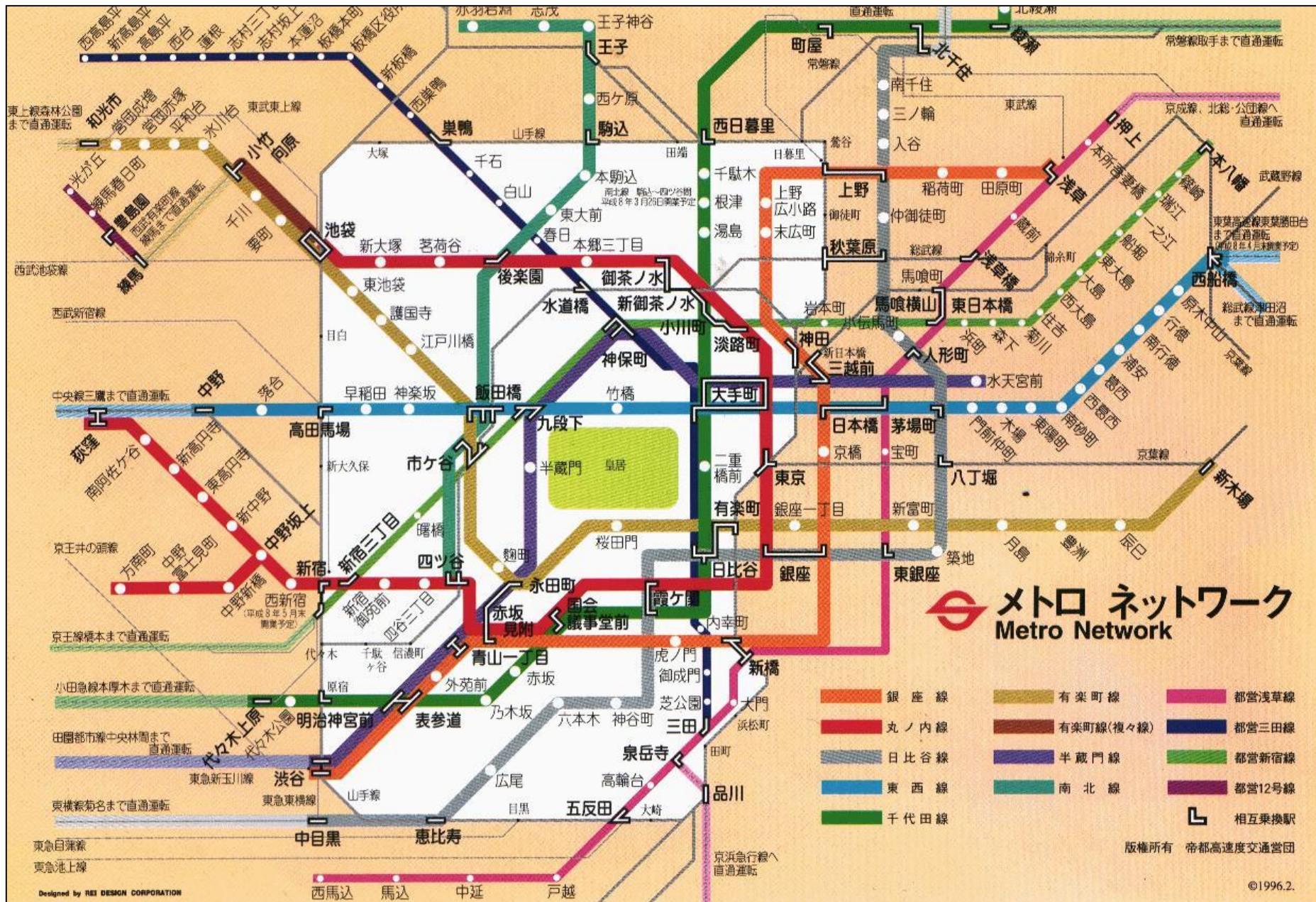


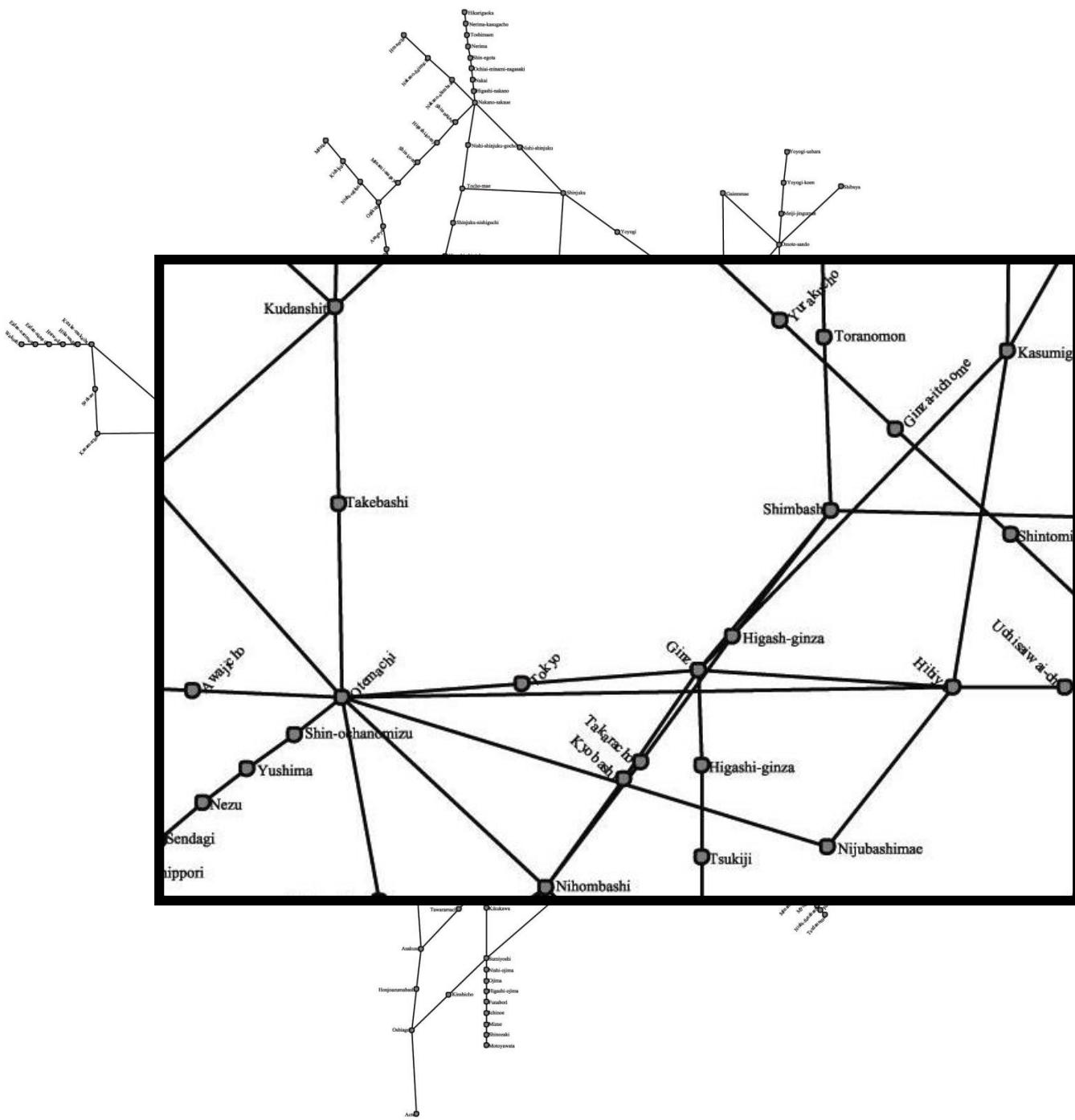
メトロ ネットワーク Metro Network

銀座線	有楽町線
丸ノ内線	都営浅草線
日比谷線	有楽町線(複々線)
東西線	都営三田線
南北線	都営新宿線
千代田線	都営12号線
相互乗換駅	

版權所有 帝都高速度交通營團

©1996.2.

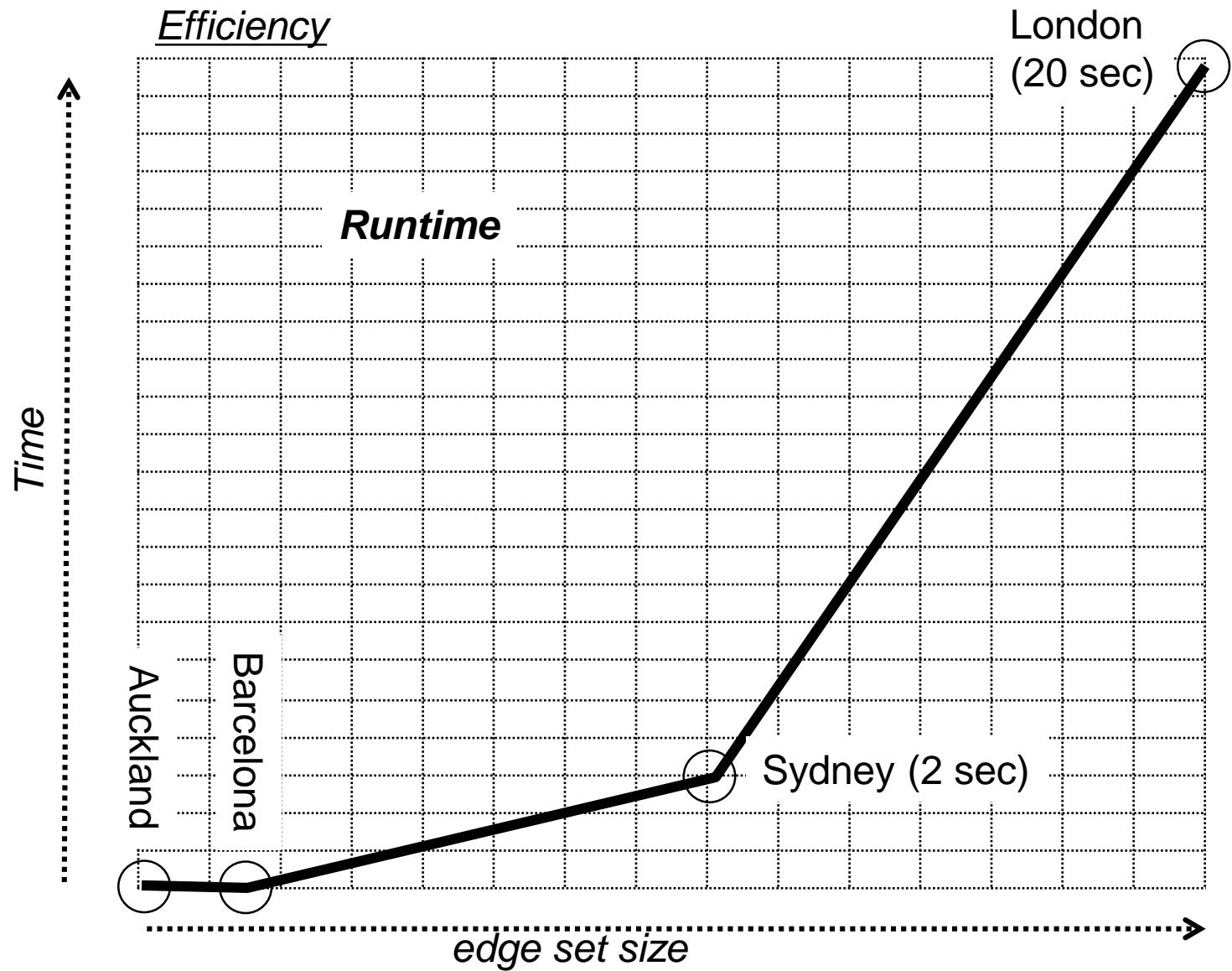




Effectiveness

- The force directed method is OK for the metro map problem
- but not wonderfully effective.

Efficiency



This energy based method is not computationally *efficient*.

Conclusion about metro maps:

- Simple force directed methods for metro maps “do not scale”
 - Runtime is too large
 - Some tangles in some maps

Elegance	Yes!
Effectiveness	Maybe
Efficiency	No

*Poor performance
when data size is
large*

4. FADE (Barnes-Hutt)

Problem: Simple spring methods are too slow for huge graphs

1. \mathbf{p}_u = some initial position for each node u ;
2. Repeat
 - 2.1 $\mathbf{F}_u := \mathbf{0}$ for each node u ;
 - 2.2 For each pair u, v of nodes
 - 2.2.1 calculate the force f_{uv} between u and v ;
 - 2.2.2 $\mathbf{F}_u += f_{uv}$;
 - 2.2.3 $\mathbf{F}_v += f_{uv}$;
 - 2.3 For each node u , $\mathbf{p}_u += \epsilon \mathbf{F}_u$;
- Until \mathbf{p}_u converges for all u ;

Computing the forces takes quadratic time

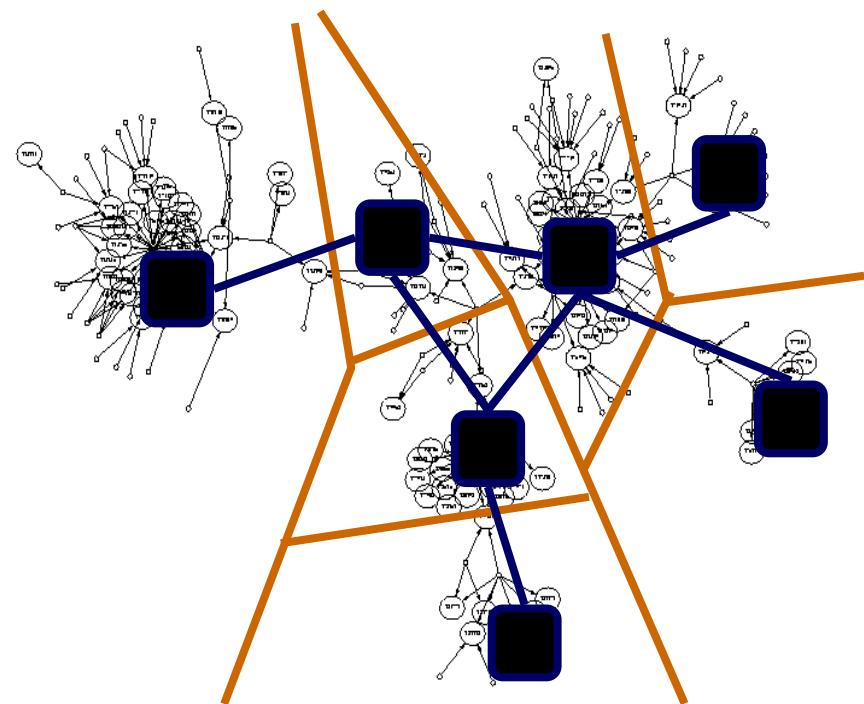
Multilevel approaches

We can use

1. A spring method, and
2. A clustering method to give a cluster hierarchy for the graph
 - The clustering could be geometric or combinatorial

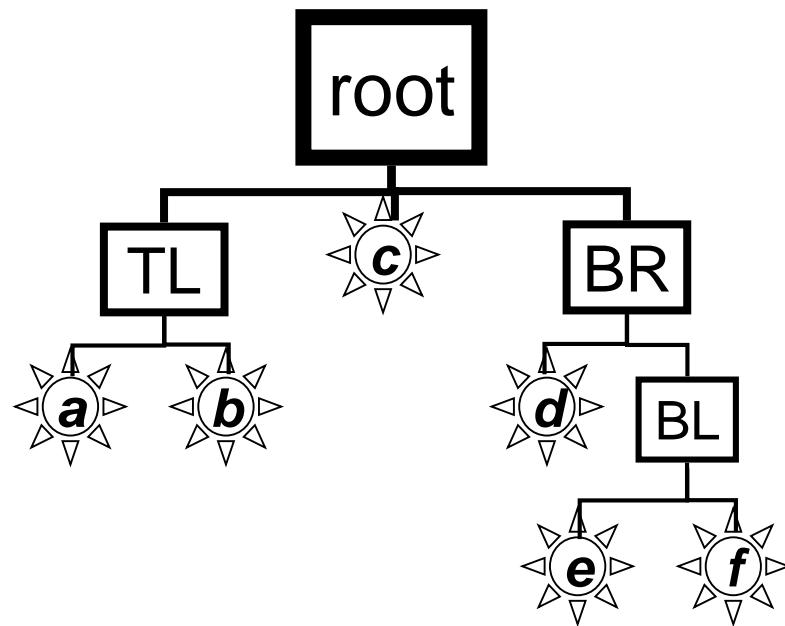
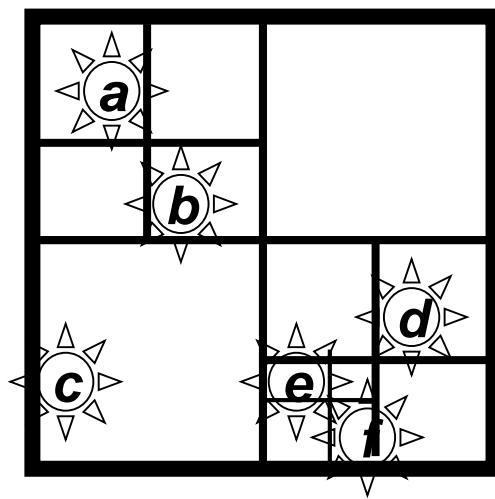
Use divide&conquer on the cluster hierarchy:

- First apply the spring method at a higher level in the hierarchy, then at a lower level



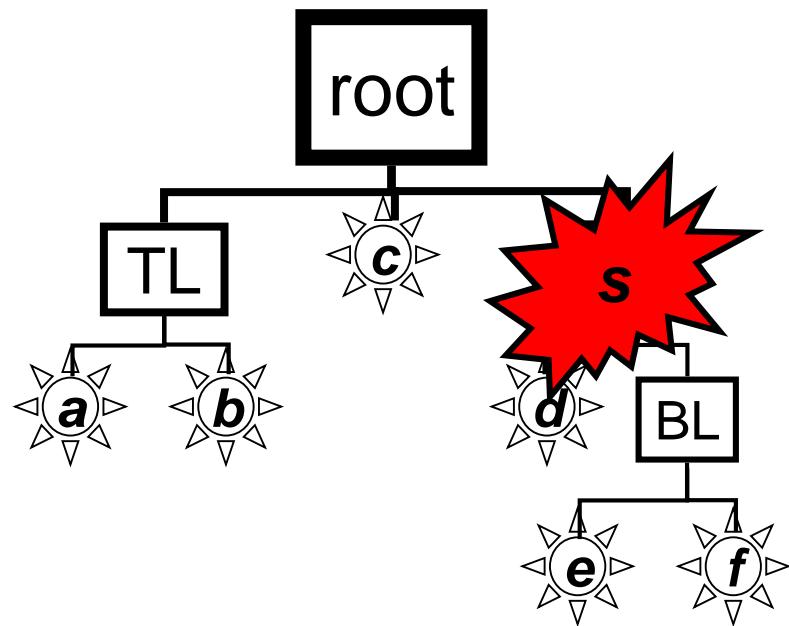
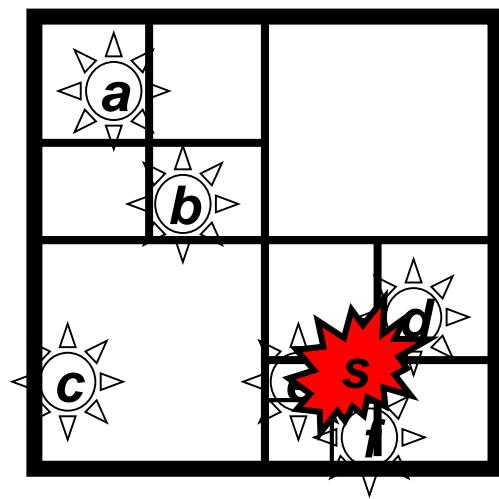
Barnes-Hutt method (1986)

- A method of computing configurations of stars.
 - Use an octtree to cluster the stars
 - Use the forces between the clusters to approximate the forces between individual stars.

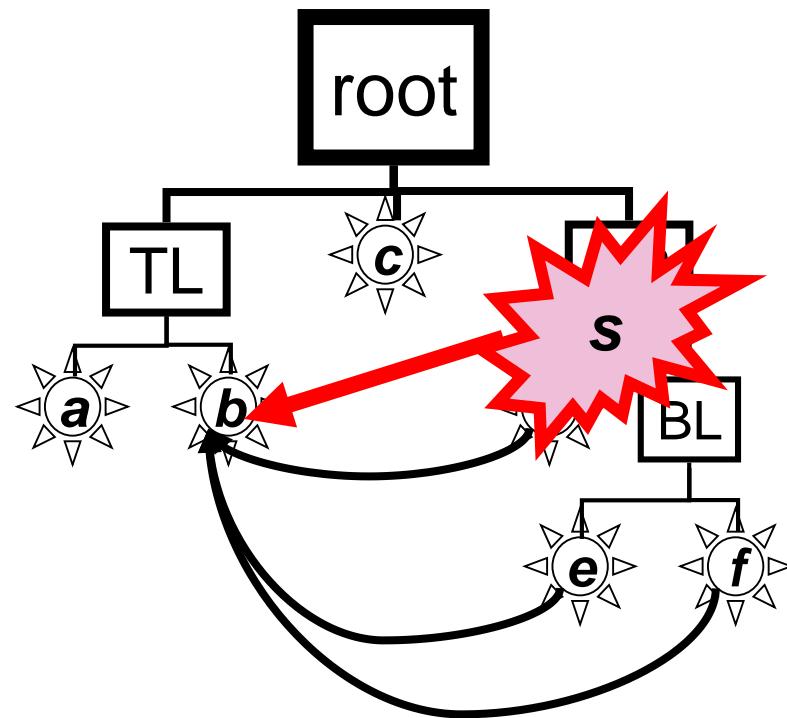
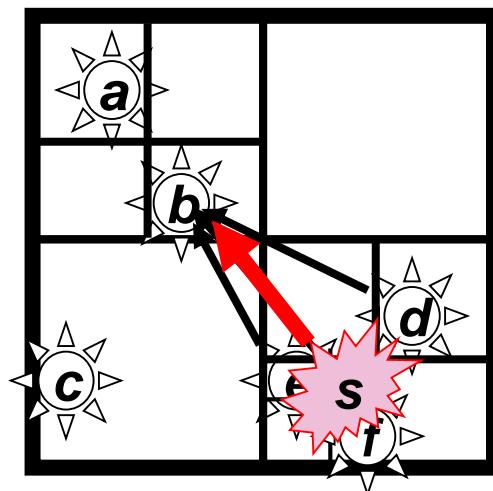


FADE

The contents of a subtree of can be approximated by a mass at the centroid.



The force that the subtree s exerts on the star b can approximate the sum of the forces that the nodes in s exert on b .



To compute the force on star x , we proceed from the root toward the leaves.

ComputeForce(star x ; treenode t)

If the approximation is good

then return the approximation;

else return $\sum_s \text{ComputeForce}(x, s)$, where the sum is over all children s of t .

A simple method can be used to determine whether the approximation is good; it depends on the *mass of nodes* and the *distance* between x and s .

$$\frac{w(t)}{d(x,t)} < c,$$

where $w(t)$ is the width of t ; $d(x, t)$ is the distance between x and t , and c is a constant.

The Barnes-Hutt method is much faster than the usual spring algorithm.

1. p_x = some initial position for each star x ;
2. Repeat

- 2.1 Build the quadtree;
- 2.2 Foreach star x

ComputeForce($x, root$);

- 2.3 Foreach star x : $p_x += \epsilon F_x$;

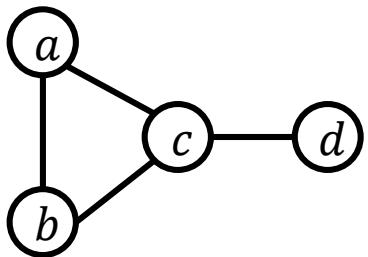
Until p_x converges for all x ;

*In practice, computing all
the forces takes
 $O(n \log n)$ time*

FADE method for graph drawing:

1. **Model**: Use springs and electrical forces
2. **Algorithm**: Apply Barnes-Hutt to the graph

5. Spectral layout



Adjacency matrix

	a	b	c	d
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

Degree matrix

	a	b	c	d
a	2	0	0	0
b	0	2	0	0
c	0	0	3	0
d	0	0	0	1

Laplacian matrix

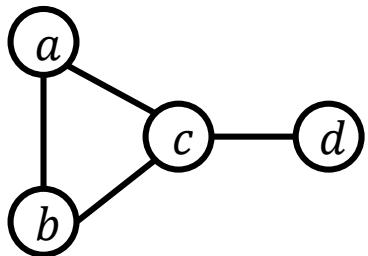
$$\mathbf{L} = \mathbf{D} - \mathbf{M}$$

	a	b	c	d
a	2	-1	-1	0
b	-1	2	-1	0
c	-1	-1	3	-1
d	0	0	-1	1

The Laplacian is a nice matrix, and its eigenvalues are nice:

- All eigenvalues of \mathbf{L} are real and non-negative.
- We put eigenvalues in non-decreasing order, so the eigenvalues of Laplacian \mathbf{L} are $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$.
- The sequence $\lambda_1, \lambda_2, \dots, \lambda_n$ is the spectrum of \mathbf{L} (and the graph G).
- The spectrum of a graph defines many of its properties.
- The study of graphs and their spectra is Spectral Graph Theory

Note: all rowsums of a Laplacian L are zero.



$$L =$$

	a	b	c	d
a	2	-1	-1	0
b	-1	2	-1	0
c	-1	-1	3	-1
d	0	0	-1	1

Thus $L\tilde{\mathbf{1}} = \mathbf{0}$ where $\tilde{\mathbf{1}} = [1, 1, \dots, 1]^T$.

Thus $L\tilde{\mathbf{1}} = \mathbf{0}\tilde{\mathbf{1}}$, that is $\lambda = 0$ is an eigenvalue of L with corresponding eigenvector $\tilde{\mathbf{1}}$.

Spectral layout (the simplest form)

2. Algorithm

Input: a graph $G = (V, E)$ with n vertices.

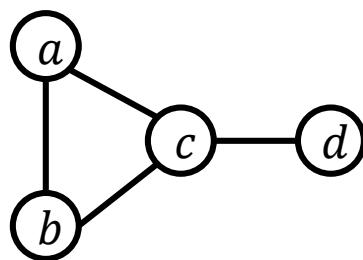
Output: a location $p(i) \in R^2$ for each vertex $i \in V$.

1. Compute the Laplacian L of G .
2. Compute the first two nonzero eigenvalues λ_2, λ_3 of L
3. Compute their corresponding eigenvectors

$$x_2 = [x_{21}, x_{22}, \dots, x_{2n}]^T \text{ and } x_3 = [x_{31}, x_{32}, \dots, x_{3n}]^T$$

4. For each vertex $i \in V$, $p(i) = (x_{2i}, x_{3i})$.

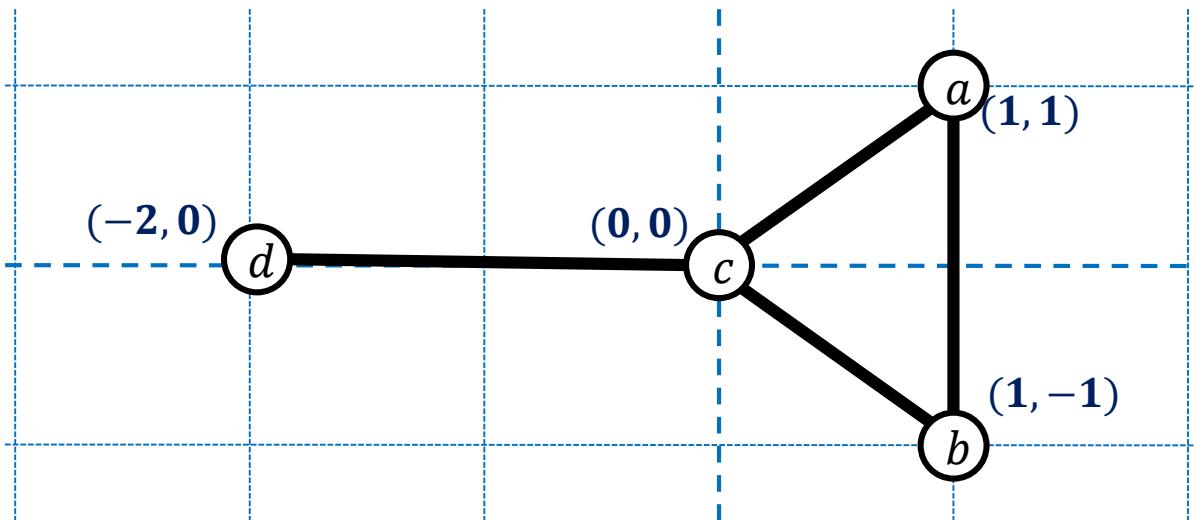
Example:



$$L = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

- Eigenvalues of L are: $\lambda_1 = 0$, $\lambda_2 = 1$, $\lambda_3 = 3$, $\lambda_4 = 4$.

- Eigenvectors of L are: $u_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$, $u_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ -2 \end{bmatrix}$, $u_3 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$, $u_4 = \begin{bmatrix} -1 \\ -1 \\ 3 \\ -1 \end{bmatrix}$.
- Thus $p(a) = (1, 1)$, $p(b) = (1, -1)$, $p(c) = (0, 0)$, $p(d) = (-2, 0)$.



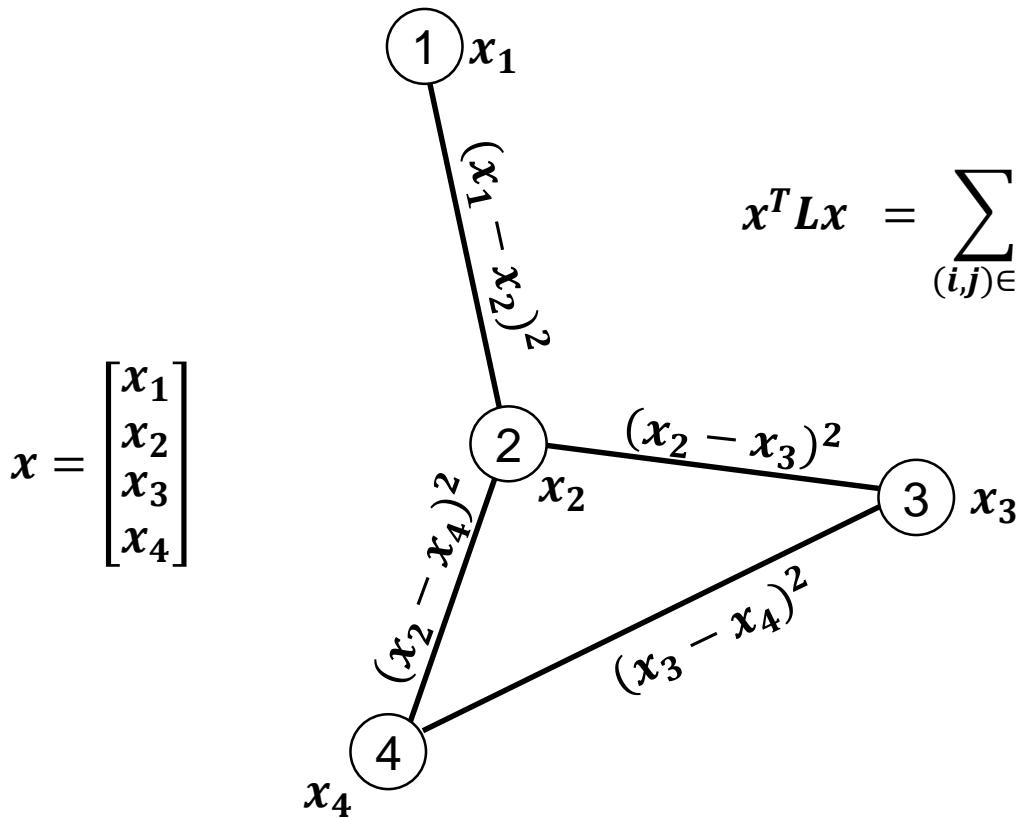
1. The model

- Spectral layout is a force-directed algorithm (sort-of)

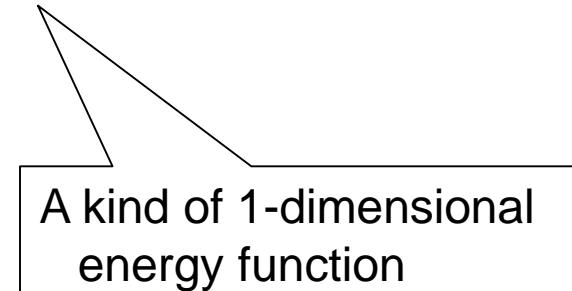
Easy Lemma

If \mathbf{L} is the Laplacian of a graph $G = (V, E)$, and x is a vector, then

$$\mathbf{x}^T \mathbf{L} \mathbf{x} = \sum_{(i,j) \in E} (x_i - x_j)^2.$$



$$\mathbf{x}^T \mathbf{L} \mathbf{x} = \sum_{(i,j) \in E} (x_i - x_j)^2$$



Theorem (from the Courant–Fischer–Weyl min-max principle)

Suppose that \mathbf{L} is the Laplacian of a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, and the eigenvalues of \mathbf{L} are $\mathbf{0} = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, and the eigenvector \mathbf{u}_k corresponds to λ_k . Then

$$\lambda_k = \min_{x \in U_k} (x^T \mathbf{L} x) = \min_{x \in U_k} \sum_{(i,j) \in E} (x_i - x_j)^2$$

where U_k is the set all unit vectors orthogonal to the space spanned by $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{k-1}\}$. The minimum value is achieved when $x = \mathbf{u}_k$.

What all this means :

- Eigenvalues are minimum values of the quadratic function $x^T \mathbf{L} x$, under two constraints:
 - Unit sphere constraint
 - Orthogonal to eigenvectors of smaller eigenvalues
- The function $x^T \mathbf{L} x$ is a kind of energy
- These minima occur at eigenvectors.

Theorem (from the Courant–Fischer–Weyl min-max principle)

Suppose that \mathbf{L} is the Laplacian of a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, and the eigenvalues of \mathbf{L} are $\mathbf{0} = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, and the eigenvector \mathbf{u}_k corresponds to λ_k . Then

$$\lambda_k = \min_{x \in U_k} (x^T \mathbf{L} x) = \min_{x \in U_k} \sum_{(i,j) \in E} (x_i - x_j)^2$$

where U_k is the set all unit vectors orthogonal to the space spanned by $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{k-1}\}$. The minimum value is achieved when $x = \mathbf{u}_k$.

What all this means for a Laplacian L :

- Smallest eigenvalue: $\lambda_1 = \mathbf{0}$ with eigenvector $\mathbf{u}_1 = [1, 1, \dots, 1]^T$
- Next smallest eigenvalue: $\lambda_2 = \min_{x \in U_2} \sum_{(i,j) \in E} (x_i - x_j)^2$
 - The minimum is over all unit vectors orthogonal to $\mathbf{u}_1 = [1, 1, \dots, 1]^T$
 - The minimum is at $x = \mathbf{u}_2$, the second eigenvector
- Next smallest eigenvalue: $\lambda_3 = \min_{x \in U_3} \sum_{(i,j) \in E} (x_i - x_j)^2$
 - The minimum is over all unit vectors orthogonal to $\mathbf{u}_1 = [1, 1, \dots, 1]^T$ and \mathbf{u}_2
 - The minimum is at $x = \mathbf{u}_3$, the third eigenvector

Theorem (from the Courant–Fischer–Weyl min-max principle)

Suppose that \mathbf{L} is the Laplacian of a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, and the eigenvalues of \mathbf{L} are $\mathbf{0} = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, and the eigenvector \mathbf{u}_k corresponds to λ_k . Then

$$\lambda_k = \min_{x \in U_k} (x^T \mathbf{L} x) = \min_{x \in U_k} \sum_{(i,j) \in E} (x_i - x_j)^2$$

where U_k is the set all unit vectors orthogonal to the space spanned by $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{k-1}\}$. The minimum value is achieved when $x = \mathbf{u}_k$.

What all this means for a Laplacian L :

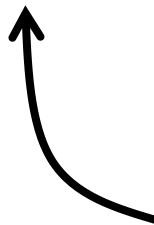
- Second smallest eigenvalue λ_2 minimizes a kind of energy function
- Third smallest eigenvalue: λ_3 minimizes a kind of energy function

Theorem (from the Courant–Fischer–Weyl min-max principle)

Suppose that \mathbf{L} is the Laplacian of a graph $G = (V, E)$, and the eigenvalues of \mathbf{L} are $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, and the eigenvector \mathbf{u}_k corresponds to λ_k . Then

$$\lambda_k = \min_{x \in U_k} (x^T \mathbf{L} x) = \min_{x \in U_k} \sum_{(i,j) \in E} (x_i - x_j)^2$$

where U_k is the set all unit vectors orthogonal to the space spanned by $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{k-1}\}$. The minimum value is achieved when $x = \mathbf{u}_k$.



Thus (roughly speaking) the eigenvectors achieve a constrained minimum energy in the edges

Remark:

There is mathematical evidence that spectral layout is good (in theory)

There is some mathematical evidence that spectral layout is good (in theory):

Spectral layout

- Should give faithful clusters
 - The layout gives a relaxation of the ratio-cut problem
- Should give faithful commute distances
 - Random walks are preserved

But in practice the simple spectral layout is not as good as it is in theory.

Spectral layout

Elegance:

- Easy to understand
- Can be coded very quickly using a numerical package, even if you don't understand eigenvalues

Efficiency

- As fast as the best algorithm that you can find to compute the first two nonzero eigenvalues of a Laplacian.
- Scales very well.

Effectiveness

- On real-world graphs it is not so good.
- It works well on mesh-like graphs.

Summary: Energy based methods

Advantages:

- Elegant: Many methods are relatively easy to understand and easy to code
- Good flexibility; supports some constraints
- Heuristic improvements easily added
- Smooth evolution 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
- Results are OK

Disadvantages

- High-quality results need high runtime.
- Results are OK but not brilliant.
- Few guarantees on the quality of the drawings produced.
- Difficult to extend to orthogonal & polyline drawings.

Announcement

- Assignments 1 & 2: will be posted on Week 4
- Group assignment for Assignment 2
 - Week 4: Form a group (upto 6 students)
 - Week 5: Final assignment