V3 Matrix algorithms and graph partitioning

- Dividing networks into clusters

- Graph partitioning

- The Kernighan-Lin algorithm

- Spectral partitioning

Motivation: Cellular networks are modular!

Adam Arkin / UC Berkeley

Modularity is one way to reconcile the seemingly incompatible objectives of **complexity** and **evolvability**.

Modularity has been shown to underlie **biological function** e.g. at the levels of transcription and embryonic development.

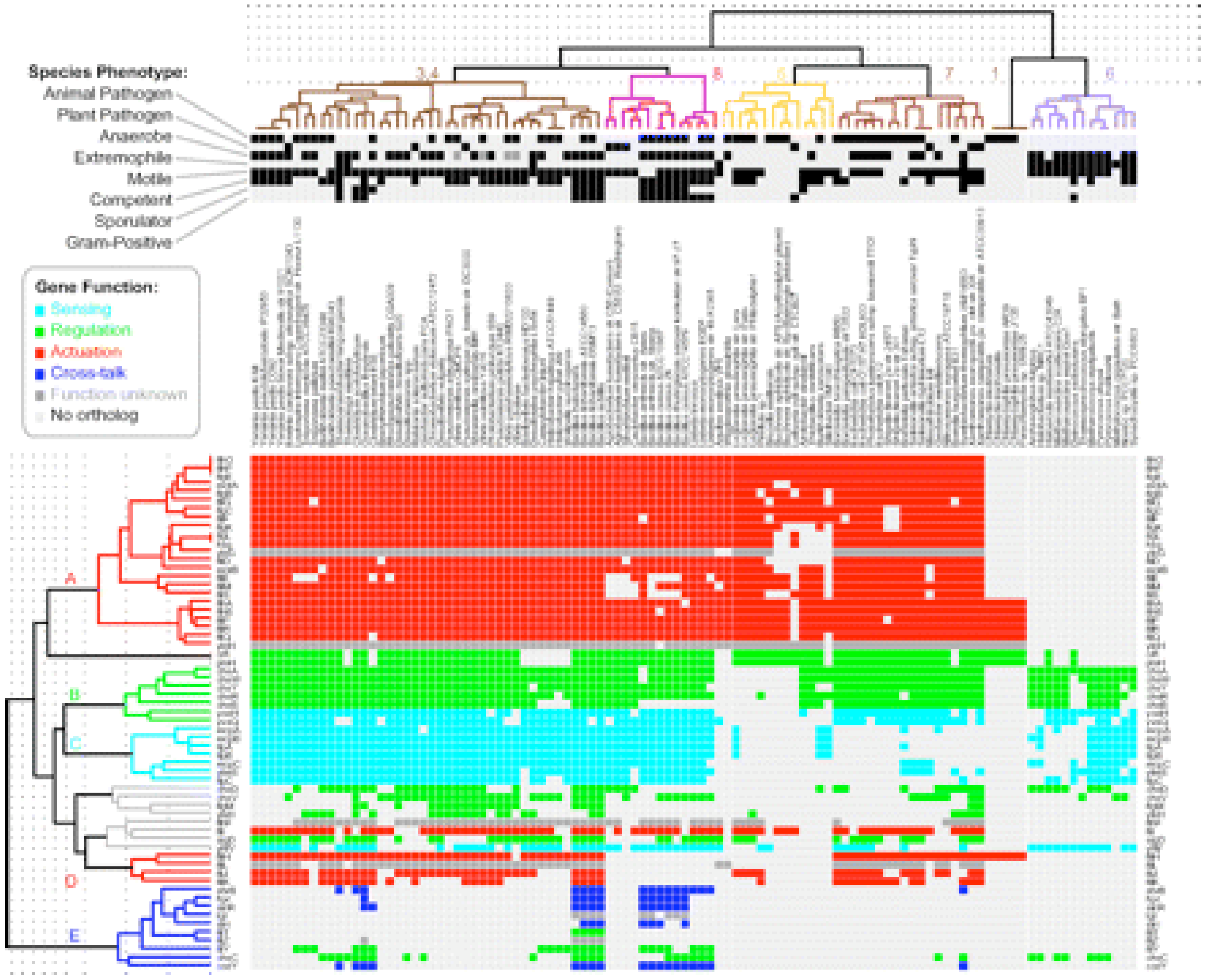
Over half of all functional modules (in the form of transcriptional modules, protein complexes, and metabolic pathways) have coevolving components.

Modularity is **hierarchical**.

Singh … Arkin PNAS (2008) 105, 7500-7505

Evolutionary modules in chemotaxis

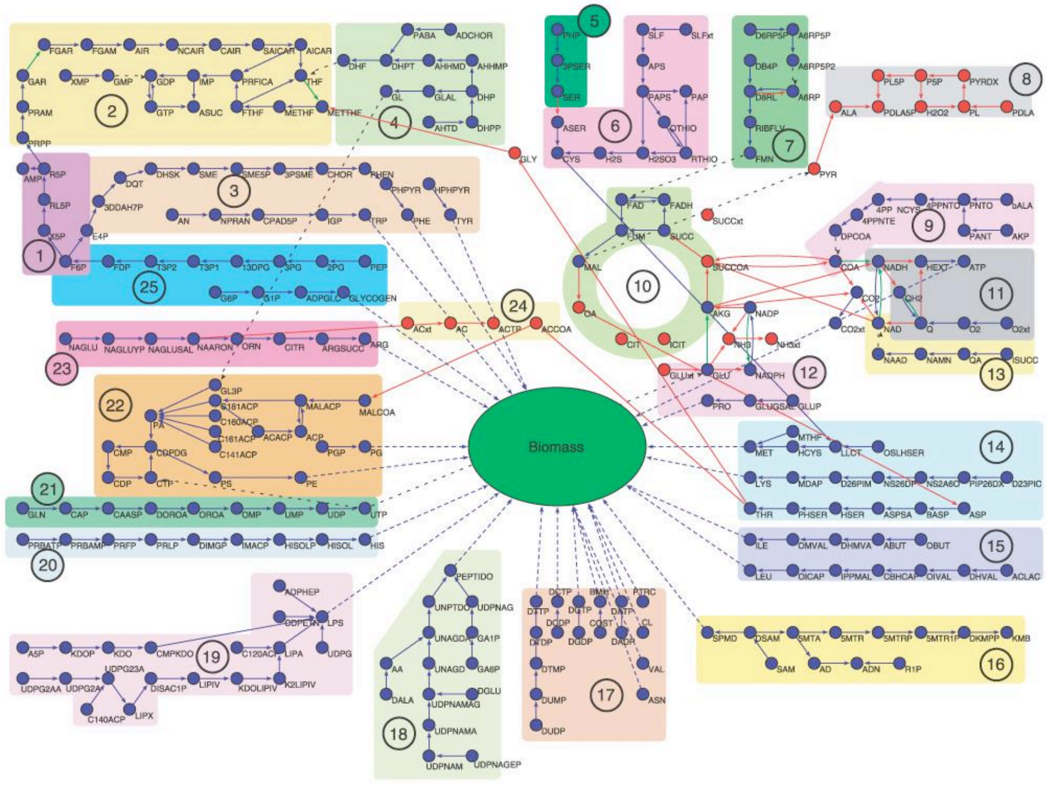
Orthologs of 61 *B. subtilis* chemotaxis genes from 207 microbial species. The resulting gene content matrix was hierarchically clustered along both genes (rows) and species (columns).
Genes were then colored according to which dynamic-control role they occupy in the network.
The clustering reveals that genes group into 5 statistically significant evolutionary modules (A–E).
(*i*) flagellar genes (*flg*, *fli*, *flh*) are conserved among motile bacteria but not among motile Archaea;
(*ii*) the full complement of signal transducers (*mcp*, *tlp*) and regulators (*che*) is absent in many intracellular pathogens.



Singh … Arkin PNAS (2008) 105, 7500-7505

FBA-optimized network on glutamate-rich substrate

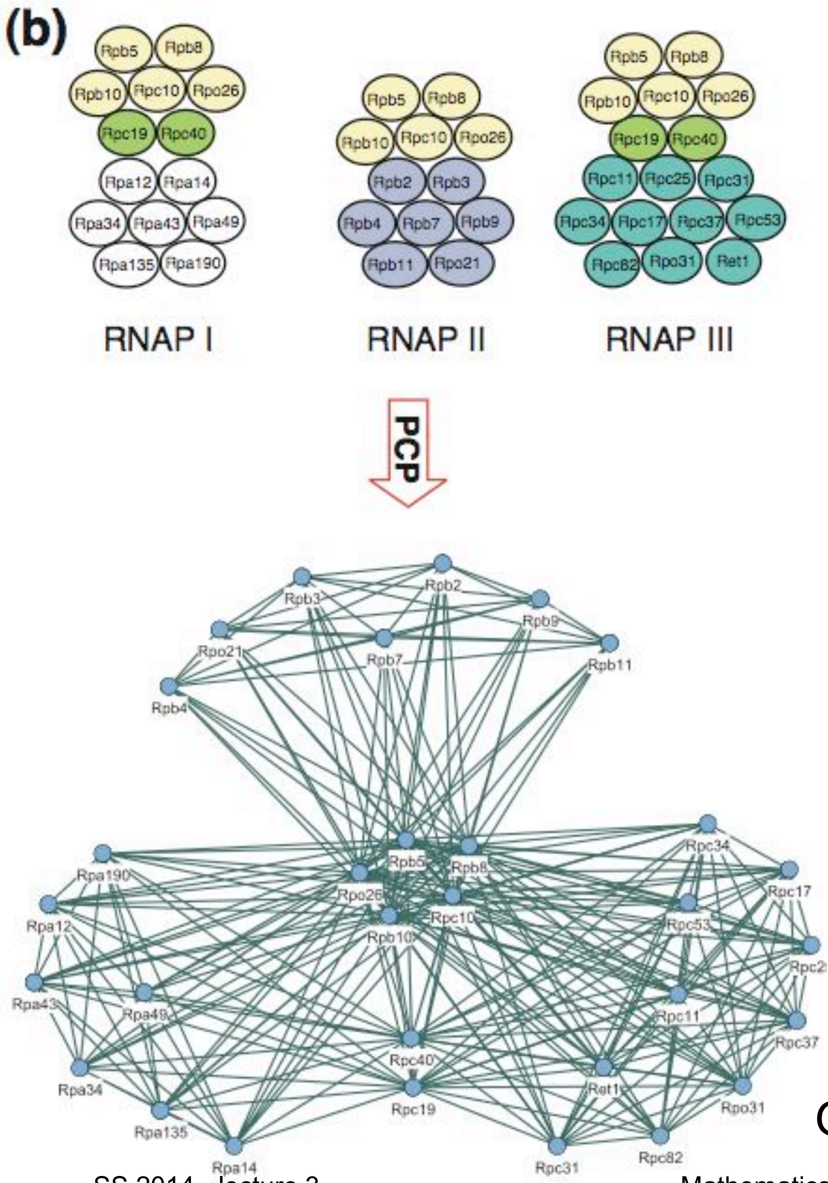
High-flux backbone for FBA-optimized metabolic network of *E. coli* on a glutamate-rich substrate. **Metabolites** (vertices) coloured blue have at least one neighbour in common in glutamate- and succinate-rich substrates, and those coloured red have none. **Reactions** (lines) are coloured blue if they are identical in glutamate- and succinate-rich substrates, green if a different reaction connects the same neighbour pair, and red if this is a new neighbour pair. Black dotted lines indicate where the disconnected pathways, for example, folate biosynthesis, would connect to the cluster through a link that is not part of the HFB. Thus, the red nodes and links highlight the predicted changes in the HFB when shifting *E. coli* from glutamate- to succinate-rich media. Dashed lines indicate links to the biomass growth reaction.

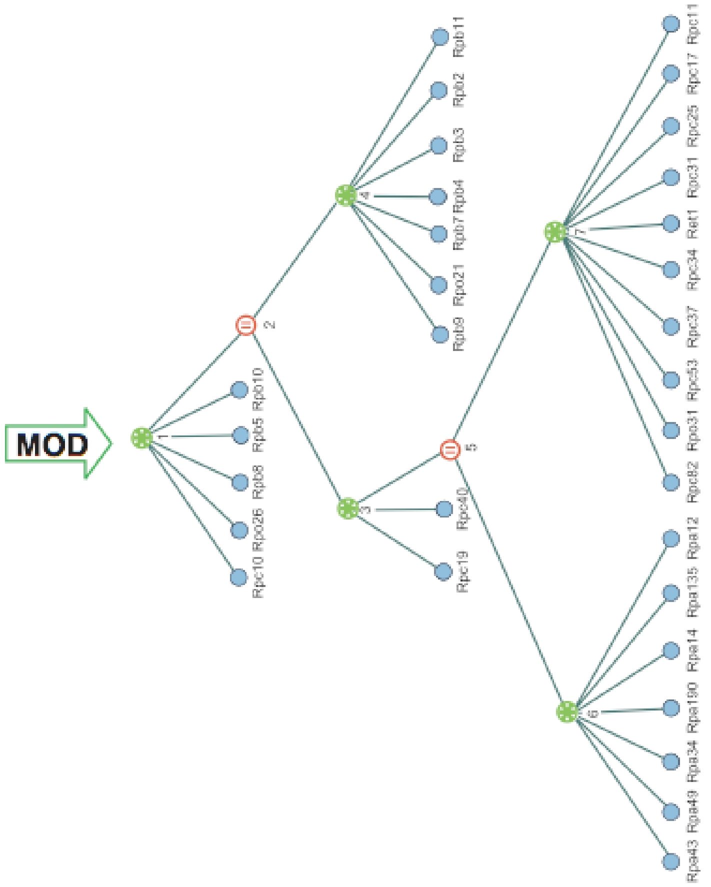


(1) Pentose Phospate
(2) Purine Biosynthesis
(3) Aromatic Amino Acids
(4) Folate Biosynthesis
(5) Serine Biosynthesis
(6) Cysteine Biosynthesis
(7) Riboflavin Biosynthesis
(8) Vitamin B6 Biosynthesis
(9) Coenzyme A Biosynthesis
(10) TCA Cycle
(11) Respiration
(12) Glutamate Biosynthesis
(13) NAD Biosynthesis
(14) Threonine, Lysine and Methionine Biosynthesis
(15) Branched Chain Amino Acid Biosynthesis
(16) Spermidine Biosynthesis
(17) Salvage Pathways
(18) Murein Biosynthesis
(19) Cell Envelope Biosynthesis
(20) Histidine Biosynthesis
(21) Pyrimidine Biosynthesis
(22) Membrane Lipid Biosynthesis
(23) Arginine Biosynthesis
(24) Pyruvate Metabolism
(25) Glycolysis

Almaar et al., Nature 427, 839 (2004)

RNA polymerases I, II and III





Again: modular decompositon easier to comprehend than graph

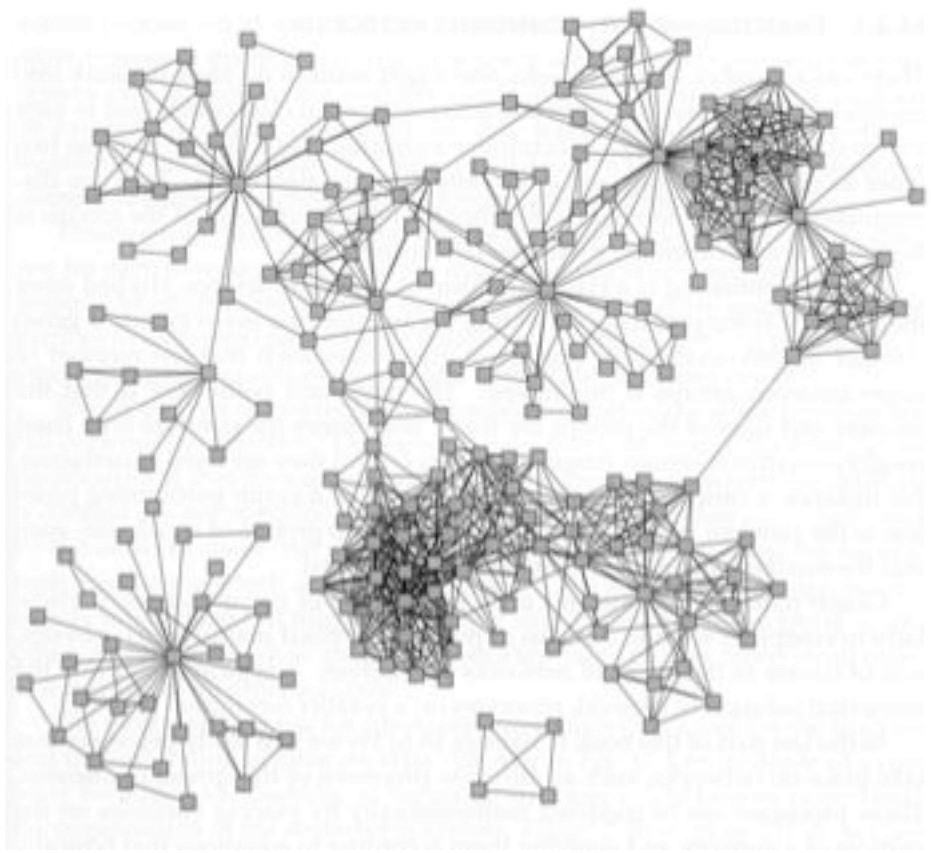
Gagneur et al. Genome Biology 5, R57 (2004)

Dividing networks into clusters

We like to divide the vertices of a graph so that vertices in one group have many edges to other vertices inside the same group and only a few edges to vertices in other groups.

Network of **Co-authorships** in a university department.
Vertices are scientists and edges link pairs of scientists who have co-authored scientific publications.

The network has clear clusters or „communities“ that likely reflect divisions of interests and research groups.



Graph partitioning

Graph partitioning and **community detection** are distinguished from one another by whether the number and size of the groups is fixed by the experimenter or whether it is unspecified.

Graph partitioning is a **classic problem** in computer science, studied since the 1960s.
It is the problem of dividing the vertices of a network into a given number of non-overlapping groups of given sizes such that the number of edges between groups is minimized.

One important application is the optimal distribution of vertices onto cores of a parallel computer in order to mimimize the amount of communication required when solving e.g. a system of coupled equations.

Community detection

In **community detection**, the number and size of the groups into which the network is divided are not specified by the experimenter but by the network itself.

The goal of community detection is to find the natural fault lines along which a network separates.

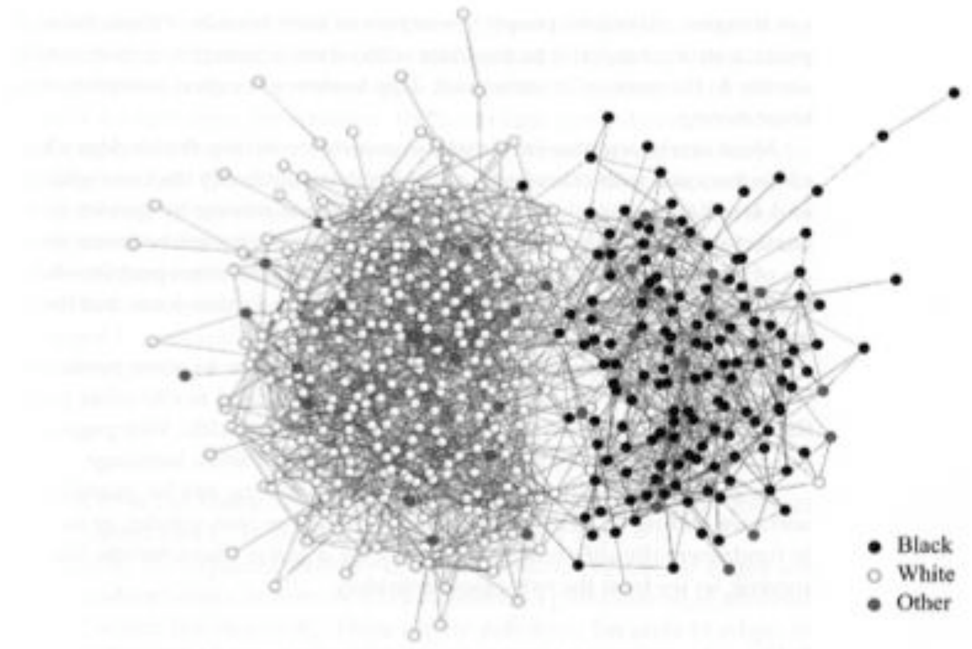
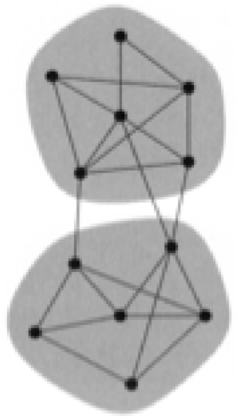


Figure 7.10: Friendship network at a US high school. The vertices in this network represent 470 students at a US high school (ages 14 to 18 years). The vertices are color coded by race as indicated in the key. Data from the National Longitudinal Study of Adolescent Health [34, 314].

Algorithms for graph partitioning

Why is partitioning hard?

The simplest graph partitioning problem is the division of a network into just 2 parts. This is sometimes called **graph bisection**.



If we can divide a network into 2 parts, we can also divide it further by dividing one or both of these parts …

The **graph bisection problem** is the problem of dividing the vertices of a network into 2 non-overlapping groups of given sizes such that the **number of edges** running **between** vertices in **different groups is minimized**.

The number of edges between groups is called the **cut size**.

Thus, one could simply look through all possible divisions of the network into 2 parts and choose the one with smallest cut size.

Algorithms for graph partitioning

But this exhaustive search is prohibitively expensive!

Given a network of n vertices. There are $\frac{n!}{n_1!n_2!}$ different ways of dividing it into 2 groups of n_1 and n_2 vertices.

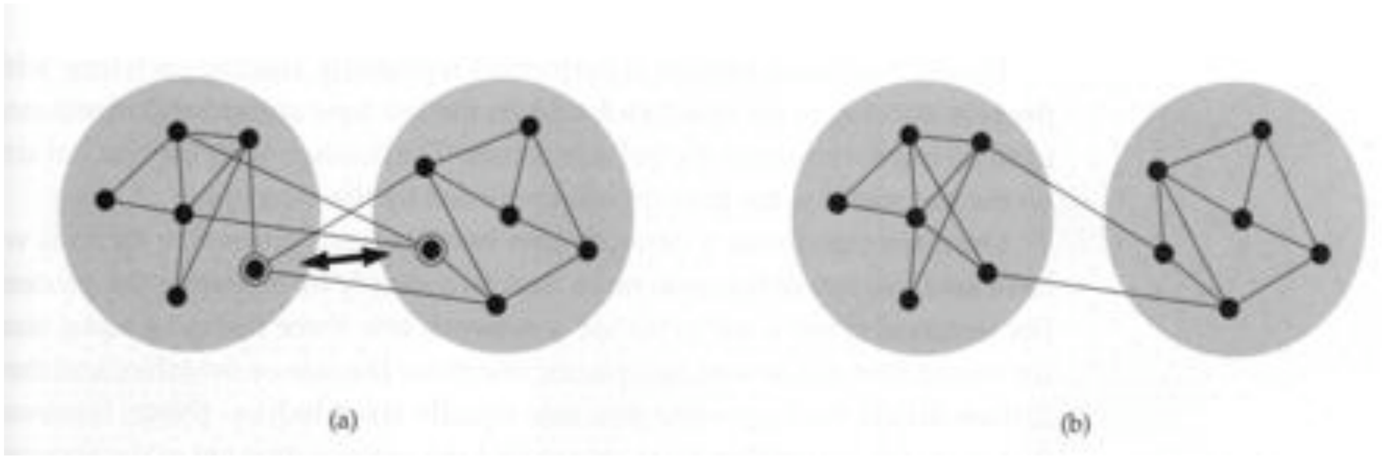
The amount of time to look through all these divisions will go up roughly exponentially with the size of the system.

Only values of up to $n = 30$ are feasible with current computers.

In computer science, either an algorithm can be clever and run quickly, but will fail to find the optimal answer in some (and perhaps most) cases, or it will always find the optimal answer, but takes an impractical length of time to do it.

The Kernighan-Lin algorithm

This algorithm proposed by Brian Kernighan and Shen Lin in 1970 is one of the simplest and best known heuristic algorithms for the graph bisection problem.
(Kernighan is also one of the developers of the C language).



(a) The algorithm starts with any division of the vertices of a network into two groups (shaded) and then searches for pairs of vertices, such as the pair highlighted here, whose interchange would reduce the cut size between the groups.
(b) The same network after interchange of the 2 vertices.

The Kernighan-Lin algorithm

(1) Divide the vertices of a given network into 2 groups (e.g. randomly)

(2) For each pair *(i,j)* of vertices, where *i* belongs to the first group and *j* to the second group, calculate how much the cut size between the groups would change if *i* and *j* were interchanged between the groups.

(3) Find the pair that reduces the cut size by the largest amount.

If no pair reduces it, find the pair that increases it by the smallest amount.

Repeat this process, but with the important restriction that each vertex in the network can only be moved once.

Stop when there is no pair of vertices left that can be swapped.

The Kernighan-Lin algorithm (II)

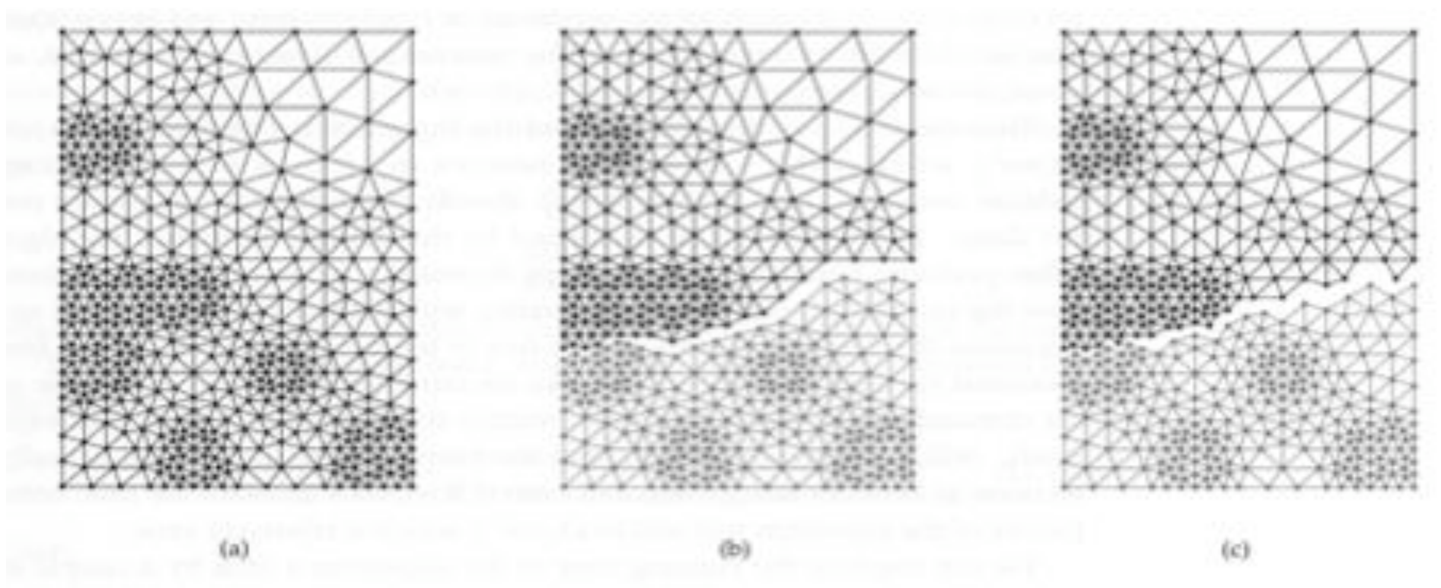
(3) Go back through every state that the network passed through during the swapping procedure and choose among them the state in which the cut size takes its smallest value.

(4) Perform this entire process repeatedly, starting each time with the best division of the network found in the last round.

(5) Stop when no improvement on the cut size occurs.

Note that if the initial assignment of vertices to group is done randomly, the Kernighan-Lin algorithm may give different answers when it is run twice on the same network.

The Kernighan-Lin algorithm (II)



(a) A mesh network of 547 vertices of the kind commonly used in finite element analysis.

(b) The best division found by the Kernighan-Lin algorithm when the task is to split the network into 2 groups of almost equal size.

This division involves cutting 40 edges in this mesh network and gives parts of 273 and 274 vertices.

(c) The best division found by spectral partitioning (second half of V3).

Runtime of the Kernighan-Lin algorithm

The number of swaps performed during one round of the algorithm is equal to the smaller of the sizes of the two groups $\in [0, n / 2]$.

$\rightarrow$ in the worst case, there are $O(n)$ swaps.

For each swap, we have to examine all pairs of vertices in different groups to determine how the cut size would be affected if the pair was swapped.

In the worst case, there are $n / 2 \times n / 2 = n^2 / 4$ such pairs, which is $O(n^2)$.

Runtime of the Kernighan-Lin algorithm (ii)

When a vertex *i* moves from one group to the other group, any edges connecting it to vertices in its current group become edges between groups after the swap.

Let us suppose that are k_i^{same} such edges.

Similarly, any edges that *i* has to vertices in the other group, (say k_i^{other} ones) become within-group edges after the swap.

There is one exception. If *i* is being swapped with vertex *j* and they are connected by an edge, then the edge is still between the groups after the swap

→ the change in the cut size due to the movement of *i* is $k_i^{other} - k_i^{same} – A_{ij}$

A similar expression applies for vertex *j*.

→ the total change in cut size due to the swap is $k_i^{other} - k_i^{same} + k_j^{other} - k_j^{same} – 2A_{ij}$

Runtime of the Kernighan-Lin algorithm (iii)

For a network stored in adjacency list form, the evaluation of this expression involves running through all the neighbors of i and j in turn, and hence takes time on the order of the average degree in the network, or $O\,(m/n)$ with m edges in the network.

$\rightarrow$ the total running time is $O\,(\,n \times n^2 \times m/n\,) = O(mn^2)$

On a sparse network with $m \propto n$, this is $O(n^3)$

On a dense network (with $m \rightarrow \frac{n(n-1)}{2}$) , this is $O(n^4)$

This time still needs to be multiplied by the number of rounds the algorithm is run before the cut size stops decreasing.
For networks up to a few 1000 of vertices, this number may be between 5 and 10.

Spectral partitioning

This method was presented by Fiedler (1973) and makes use of the matrix properties of the graph **Laplacian**.

Again we will apply this algorithm to the graph bisection problem.

Given a network of *n* vertices and *m* edges and a divsion into group 1 and group 2.

We can write the **cut size** for the division as

$$R = \frac{1}{2} \sum_{\substack{i,j \text{ in} \\ \text{different} \\ \text{groups}}} A_{ij}$$

The factor ½ compensates for counting each edge twice in the sum.

Spectral partitioning

Let us define a set of quantities s_i , one for each vertex *i*, which represent the division of the network thus:

$$s_i = \begin{cases} +1 & \text{if vertex } i \text{ belongs to group 1} \\ -1 & \text{if vertex } i \text{ belongs to group 2} \end{cases}$$

Then $\frac{1}{2}(1 - s_i s_j) = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in different groups} \\ 0 & \text{if } i \text{ and } j \text{ are in the same group} \end{cases}$

This allows to rewrite the cut size as

$$R = \frac{1}{4} \sum_{i,j} A_{ij} \left(1 - s_i s_j\right)$$

where the sum now runs over all values of *i* and *j.*

Spectral partitioning

$R = \frac{1}{4}\sum_{i,j} A_{ij}\left(1 - s_i s_j\right)$ The first term in the sum is

$$\sum_{ij} A_{ij} = \sum_i k_i = \sum_i k_i {s_i}^2 = \sum_{ij} k_i \delta_{ij} s_i s_j$$

where k_i is the degree of vertex *i* , δ_{ij} is the Kronecker symbol.

We have also used the fact that $\sum_j A_{ij} = k_i$ and $s_i^2 = 1$

Substituting this back into the above equation gives

$$R = \frac{1}{4}\sum_{ij}\left(k_i \delta_{ij} - A_{ij}\right) s_i s_j = \frac{1}{4}\sum_{ij} L_{ij} s_i s_j \quad \text{or} \quad R = \frac{1}{4}\mathbf{s}^T \mathbf{L}\, \mathbf{s}$$

in matrix form, where **s** is the vector with elements s_i and
$L_{ij} = k_i\, \delta_{ij} - A_{ij}$ is the *ij*-th element of the graph **Laplacian matrix**.

Insert: graph Laplacian

The graph Laplacian is defined in analogy to the **diffusion** process.

Diffusion processes are normally treated by the **diffusion equation** $\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2}$

But one can also consider diffusion processes that take place on **networks**.

There, the rate at which the amount of substance σ_i at vertex *i* changes is determined by the flow from and to other vertices *j* that are connected to *i*.

This gives $\frac{d\sigma_i}{dt} = C \sum_j A_{ij}(\sigma_j - \sigma_i)$. By splitting the 2 terms we can write

$$\frac{d\sigma_i}{dt} = C \sum_j A_{ij}\sigma_j - C\sigma_i \sum_j A_{ij} = C \sum_j A_{ij}\sigma_j - C\sigma_i k_i$$

Insert: graph Laplacian

We can write this equation $\frac{d\sigma_i}{dt} = C \sum_j A_{ij}\sigma_j - C\sigma_i\, k_i$

in matrix form

$$\frac{d\sigma}{dt} = C(\mathbf{A} - \mathbf{D})\sigma$$

where σ is the vector with components σ_i , **A** is the adjacency matrix and **D** is a diagonal matrix with the vertex degrees on the diagonal.

It is common to define a new matrix **L** = **D** - **A** so that the above equation takes on the form of the ordinary diffusion equation

$$\frac{d\sigma}{dt} - C\,\mathbf{L}\,\sigma = 0 \qquad\qquad \frac{\partial c}{\partial t} - D\,\frac{\partial^2 c}{\partial x^2} = 0$$

Here, the Laplacian operator of the second spatial derivatives is replaced by **L**.

Insert: eigenvectors of the graph Laplacian

Consider an undirected network with *n* vertices and *m* edges.

Let us designate one end of each **edge** to be **end 1** and the other to be **end 2**.

Now let us define the $m \times n$ **incidence matrix B** with elements as follows

$$B_{ij} = \begin{cases} +1 & \text{if end 1 of edge } i \text{ is attached to vertex } j \\ -1 & \text{if end 2 of edge } i \text{ is attached to vertex } j \\ 0 & \text{otherwise} \end{cases}$$

Thus, each row of **B** has exactly one +1 and one -1 element.

Insert: eigenvectors of the graph Laplacian

Let us now consider the sum $\sum_k B_{ki} B_{kj}$ that runs over all edges for the vertex pair *i* and *j*.

If $i \neq j$, the only non-zero terms in this sum occur if both B_{ki} and B_{kj} are non-zero.

In that case, edge *k* connects vertices *i* and *j,* and the product will be -1.

For a simple network, there is at most one edge between any pair of vertices. Thus, the entire sum will be -1 if there is an edge between *i* and *j* and 0 otherwise.

If $i = j$, then the sum is $\sum_k {B_{ki}}^2$ which contributes a value of +1 for every edge connected to vertex *i,* so the whole sum is just equal to degree k_i of vertex *i*.

Insert: eigenvectors of the graph Laplacian

Thus the sum $\sum_k B_{ki} B_{kj}$ is precisely equal to an element of the Laplacian

$$\sum_k B_{ki} B_{kj} = L_{ij}$$

The diagonal elements L_{ii} are equal to the degrees k_i and the off-diagonal terms L_{ij} are -1 if there is an edge *(i,j)* and zero otherwiwse.

In matrix form, we can write $\mathbf{L} = \mathbf{B}^T \mathbf{B}$

Now let $\mathbf{v}_i$ be an eigenvector of $\mathbf{L}$ with eigenvalue α_i.

Then $\mathbf{v}_i^T \mathbf{B}^T \mathbf{B} \mathbf{v}_i = \mathbf{v}^T_i \mathbf{L} \mathbf{v}_i = \alpha_i \mathbf{v}^T_i \mathbf{v}_i = \alpha_i$

where we assumed that the eigenvector $\mathbf{v}_i$ is normalized to that its inner product with itself is 1.

Insert: eigenvectors of the graph Laplacian

Thus any eigenvalue α_i of the Laplacian is equal to $(\mathbf{v}_i^T\ \mathbf{B}^T)\ (\mathbf{B}\ \ \mathbf{v}_i)$.

This quantity is just the product of a real vector with itself. It is the sum of the squares of the real elements of that vector and hence cannot be negative.

It follows that all eigenvalues of the graph Laplacian are non-negative.
The smallest possible eigenvalue is 0.

Let us consider the vector **1** = (1,1,1,…).

If we multiply this vector by the Laplacian, the *i*-th element has the value

$$\sum_j L_{ij} \times 1 = \sum_j \left(\delta_{ij} k_i - A_{ij}\right) = k_i - \sum_j A_{ij} = k_i - k_i = 0$$

Thus the vector **1** is always an eigenvector of **L** with the smallest possible eigenvalue 0.

Back to spectral partitioning

R was the cut size for the division, i.e., the number of edges running between the 2 groups:

$$R = \frac{1}{4} \mathbf{s}^T \mathbf{L}\, \mathbf{s}$$

where **s** is the vector with elements s_i and $L_{ij} = k_i\, \delta_{ij} - A_{ij}$ is the *ij*-th element of the graph **Laplacian matrix**.

The matrix **L** specifies the structure of the network and the vector **s** defines a division of that network into groups.

Our goal is to find the vector **s** that minimizes the cut size for given **L**.

In general, this mimization problem is not easy to solve because the values s_i are restricted to +1 and -1.

Relaxation method

If the values s_i could take on any real value, we could compute the derivative, set this to zero, and find the minimum.

We will solve this problem with the **relaxation method** where we will relax these restraints.

This is one of the standard methods for finding approximate solutions of vector optimization problems.

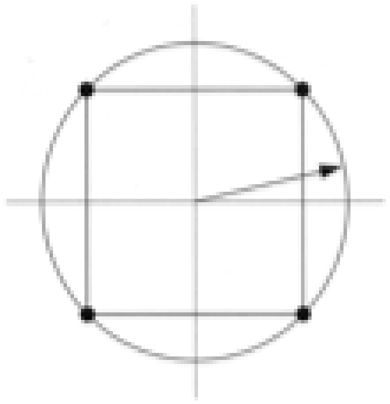
The allowed values of s_i are actually subject to 2 constraints.

First, each individual element s_i can only have the values +1 and -1.

If we regard **s** as a vector in Euclidian space then this constraint means that the vector always points to one of the 2^n corners of an *n*-dimensional hypercube centered on the origin. It always has the same length $\sqrt{n}$.

Back to spectral partitioning

Let us now relax the constraint on the vector‘s direction, so that it can point in any direction in its *n*-dimensional space.



We will however still keep its length the same.

So **s** will be allowed to take any value, subject to the constraint $|\mathbf{s}| = \sqrt{n}$ or $\sum_i {s_i}^2 = n$

The second constraint on the s_i is that the numbers of them that are equal to +1 and -1 respectively must be equal to the desired sizes of the 2 groups.

If these 2 sizes are n_1 and n_2 , this second constraint can be written as $\sum_i s_i = n_1 - n_2$ or in vector notation $\mathbf{1}^{\mathrm{T}}\mathbf{s} = n_1 - n_2$ where $\mathbf{1}^{\mathrm{T}}$ = (1,1,1, …)

We keep this second constraint unchanged.

Minimize cut size subject to constraints

Our task is therefore to mimize the cut size

$$R = \frac{1}{4} \sum_{ij} L_{ij} s_i s_j$$

subject to the 2 constraints

$$\sum_i s_i^{\,2} = n$$

$$\sum_i s_i = n_1 - n_2$$

We differentiate *R* with respect to the elements *s*$_i$ and enforce the constraints using two Lagrange multipliers which we denote α and 2μ

$$\frac{\partial}{\partial s_i}\left[\sum_{jk} L_{jk} s_j s_k + \alpha\left(n - \sum_j s_j^{\,2}\right) + 2\mu\left((n_1 - n_2) - \sum_j s_j\right)\right] = 0$$

Back to spectral partitioning

$$\frac{\partial}{\partial s_i}\left[\sum_{jk} L_{jk} s_j s_k + \alpha\left(n - \sum_j {s_j}^2\right) + 2\mu\left((n_1 - n_2) - \sum_j s_j\right)\right] = 0$$

Performing the derivatives, we then find that

$$\sum_j L_{ij} s_j + \sum_k L_{ik} s_k - 2\alpha s_i - 2\mu = 0$$

$$\sum_j L_{ij} s_j = \alpha s_i + \mu$$

or in matrix notation

$$\mathbf{L}\,\mathbf{s} = \alpha\,\mathbf{s} + \mu\,\mathbf{1}$$

We can calculate the value of μ by recalling that $\mathbf{1}$ is an eigenvector of the Laplacian with eigenvalue 0 so that $\mathbf{L} \cdot \mathbf{1} = \mathbf{1}^T L = 0$.

Relationship to eigenvectors of Laplacian

Multiplying $\mathbf{L}\,\mathbf{s} = \alpha\,\mathbf{s} + \mu\,\mathbf{1}$ from the left by $\mathbf{1}^T$ gives

$$\mathbf{1}^T\mathbf{L}\,\mathbf{s} = \alpha\mathbf{1}^T\,\mathbf{s} + \mu\,\mathbf{1}^T\mathbf{1}$$

with $\mathbf{1}^{\mathrm{T}}\mathbf{s} = n_1 - n_2$ we get

$$0 = \alpha(n_1 - n_2) + \mu\,n$$

or

$$\mu = -\frac{n_1 - n_2}{n}\alpha$$

If we define the new vector $\mathbf{x} = \mathbf{s} + \frac{\mu}{\alpha}\mathbf{1} = \mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{1}$

then $\mathbf{L}\,\mathbf{x} = \mathbf{L}\left(\mathbf{s} + \frac{\mu}{\alpha}\mathbf{1}\right) = \mathbf{L}\,\mathbf{s} = \alpha\,\mathbf{s} + \mu\,\mathbf{1} = \alpha\,\mathbf{x}$

In other words, $\mathbf{x}$ is an eigenvector of the Laplacian with eigenvalue α.

Which eigenvector should be choose?

We are still free to choose which eigenvector it is.

We should choose the one that gives the smallest value of the cut size *R*.

Notice that $\mathbf{1}^T\mathbf{x} = \mathbf{1}^T\mathbf{s} + \frac{\mu}{\alpha}\mathbf{1}^T\mathbf{1} = (n_1 - n_2) - \frac{n_1 - n_2}{n} n = 0$

Thus, **x** is orthogonal to **1**.

While **x** should be an eigenvector of **L** , it cannot be the eigenvector (1,1,1,…) that has eigenvalue 0.

Which eigenvector should we choose instead?

Choose eigenvector with smallest eigenvalue

$$R = \frac{1}{4}\mathbf{s}^T\mathbf{L}\,\mathbf{s} = \frac{1}{4}\mathbf{x}^T\mathbf{L}\,\mathbf{x} = \frac{1}{4}\alpha\,\mathbf{x}^T\mathbf{x}$$

We also have $\mathbf{x}^T\mathbf{x} = \mathbf{s}^T\mathbf{s} + \frac{\mu}{\alpha}(\mathbf{s}^T\mathbf{1} + \mathbf{1}^T\mathbf{s}) + \frac{\mu^2}{\alpha^2}\mathbf{1}^T\mathbf{1}$

$$= n - 2\frac{n_1 - n_2}{n}(n_1 - n_2) + \frac{(n_1 - n_2)^2}{n^2}n$$

$$= n - \left(\frac{(n_1 - n_2)^2}{n} + 4\frac{n_1 n_2}{n} - 4\frac{n_1 n_2}{n}\right) = n - \frac{(n_1 + n_2)^2}{n} + 4\frac{n_1 n_2}{n} = 4\frac{n_1 n_2}{n}$$

and hence

$$R = \frac{n_1 n_2}{n}\alpha$$

Thus, the cut size is proportional to the eigenvalue α.

Since our goal is to mimize *R*, we should choose **x** to be the eigenvector corresponding to the smallest allowed eigenvalue of the Laplacian.

Optimal network division

We have already shown that **x** must be orthogonal to the eigenvector **1** with the smallest eigenvalue 0.

The best thing we can do is choose **x** proportional to eigenvector $\mathbf{v}_2$ corresponding to the second lowest eigenvalue.

Finally we recover the best value for the network division **s** :

$$\mathbf{s} = \mathbf{x} + \frac{n_1 - n_2}{n}\mathbf{1}$$

or equivalently

$$s_i = x_i + \frac{n_1 - n_2}{n}$$

This gives us the optimal relaxed value of **s**.

Network partitioning with constraints

There is, however, an additional constraint that the elements of **s** should have the values of +1 and -1.

Moreover, exactly n_1 of them should be +1 and n_2 should be -1.

Thus, the values of **s** cannot exactly take on the values $s_i = x_i + \frac{n_1 - n_2}{n}$.

Let us do the best we can and choose **s** as close as possible to these ideal values subject to its contraints by making the product

$$\mathbf{s}^T \left(\boldsymbol{x} + \frac{n_1 - n_2}{n} 1 \right) = \sum_i s_i \left(x_i + \frac{n_1 - n_2}{n} \right)$$

as large as possible.

This is achieved by assigning s_i = +1 for the vertices with the larges (i.e most positive) values of $x_i + \frac{n_1 - n_2}{n}$ and thus largest values of x_i and s_i = -1 to the remaining ones.

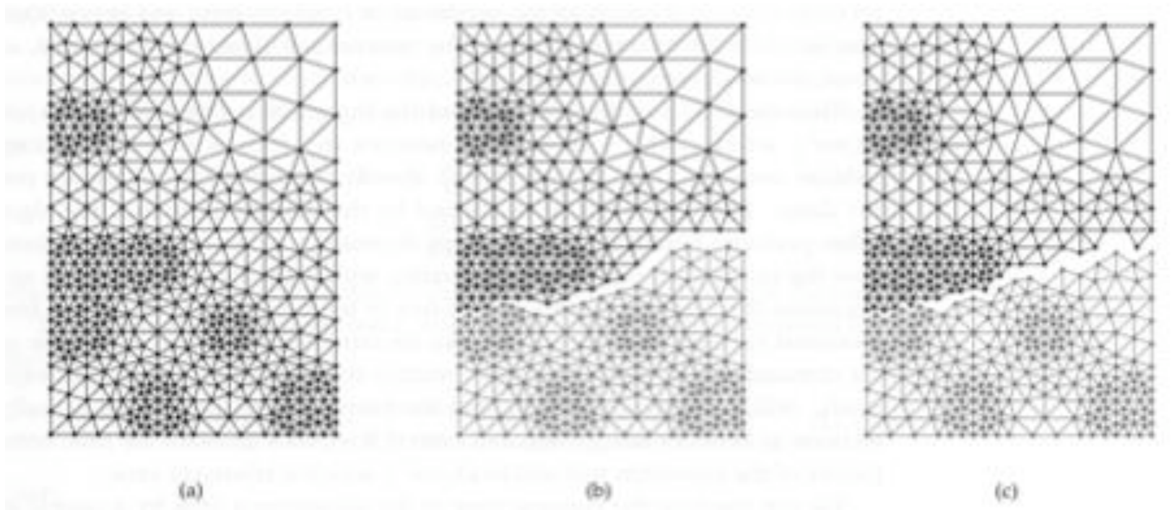
Algorithm for spectral partitioning

There is a further subtlety. It is arbitrary which group we call group 1 and which we call group 2. Thus, if the sizes of the two groups are different there are two different ways of making the split. We solve this in a pragmatic way, see below.

Thus our final algorithm is as follows:

1. Calculate the eigenvector $\mathbf{v}_2$ corresponding to the second smallest eigenvalue of the graph Laplacian.
2. Sort the elements of the eigenvector in order from largest to smallest.
3. Put the vertices corresponding to the n_1 largest elements in group 1, the rest in group 2 and calculate the cut size.
4. The put the vertices corresponding to the n_1 smallest elements in group 1, the rest in group 2 and calculate the cut size.
5. Between these 2 divisions of the network, choose the one that gives the smaller cut size.

Results and complexity of spectral partitioning



For this network, the cut-size of the Kernighan-Lin algorithm is 40, whereas spectral partitioning cuts 46 edges.

Spectral partitioning tends to find divisions of a network that have „the right general shape“.

Speed: The time-consuming part of spectral partitioning is computation of the eigenvector which takes time $O(mn)$ or $O(n^2)$ on a sparse network.
This is an order of magnitude better than the Kernighan-Lin algorithm ($O(n^3)$).