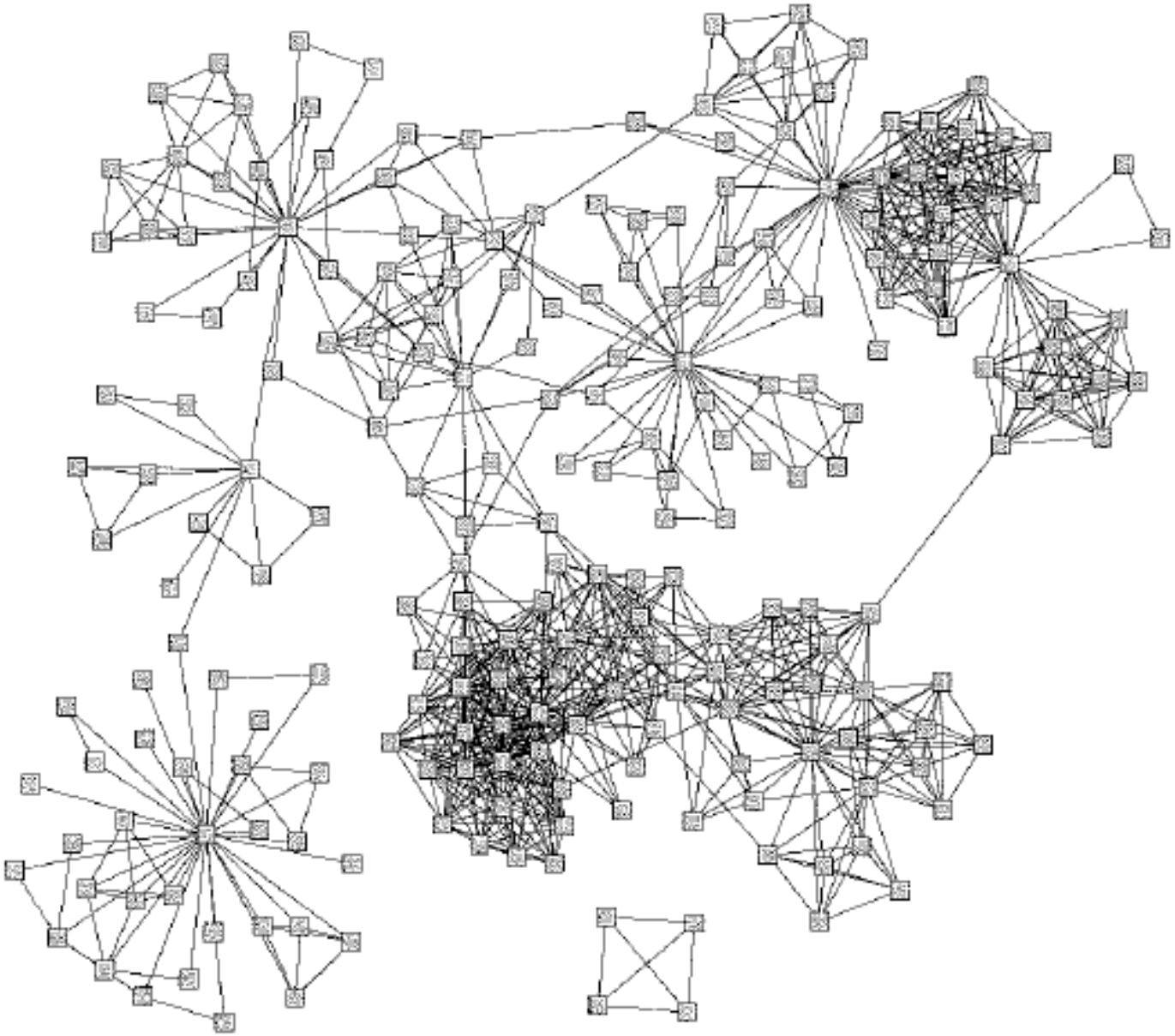
Web-graphs. Lecture 9.

Partitioning and community detection

- There are a number of reasons why one might want to divide a network into groups or clusters. Two general classes:
  - graph partitioning,
  - community detection.
- Both of these terms refer to the division of the vertices of a network into groups, clusters, or communities according to the pattern of edges in the network.
- Most commonly one divides the vertices so that the groups formed are tightly knit with many edges inside groups and only a few edges between groups.

Partitioning and community detection



From M.Newman Networks. An Introduction.

Network of coauthorships in a university department. The vertices in this network represent scientists in a university department, and edges links pairs of scientists who have coauthored scientific papers. The network has clear clusters or "community structure" presumably reflecting divisions of interests and research groups within the department.

Partitioning: problem formulation

- 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.
- A simple and prototypical example of a graph partitioning problem is the problem of dividing a network into two groups of equal size, such that the number of edges between them is minimized.

Partitioning: problem formulation

- Example: computation of different network processes on the parallel processors.
  - We divide vertices between processors.
  - Unless the network consists of totally unconnected components, some vertices on one processor are always going to have neighbors that are on the other processor.
  - The solution of their equations involves variables whose value is known only to the other processor.
  - To complete the solution those values have to be transmitted from the one processor to the other at regular intervals throughout the calculation and this is typically a slow process.
  - The time spent sending messages between processors can be the primary factor limiting the speed of calculations parallel computers, so it is important to minimize interprocessor communication.
  - One way that we do this is by minimizing the number of pairs of neighboring vertices assigned to different processors.

Community detection: problem formulation

- Community detection problems differ from graph partitioning in that the number and size of the groups into which the network divided are not specified by the experimenter
- Instead they are determined the network itself: the goal of community detection is to find the natural fault lines along which a network separates.
- The sizes of the groups are not merely unspecified but might in principle vary widely from one group to another.
- A given network might divide into a few large groups, many small ones, or a mixture of all different sizes.
- Clusters of nodes in a web graph for instance might indicate groups of related web pages. Clusters of nodes in a metabolic network might indicate functional units within the network.
- Community detection is a less well-posed problem than graph partitioning.

Partitioning goals vs Community detection goals

- Graph partitioning is typically performed as a way of dividing up a network into **smaller more manageable pieces**, for example to perform numerical calculations.
  Community detection is more often used as a tool for **understanding the structure of a network**, for shedding light on largescale patterns of connection that may not be easily visible in the raw network topology.
- In graph partitioning calculations the goal is usually to find the best division of a network, subject to certain conditions, regardless of whether any good division exists.
  With community detection, on the other hand, where the goal is normally to understand the structure of the network, there is no need to divide the network if no good division exists. Indeed if a network has no good divisions then that in itself may be a useful piece of information, and it would be perfectly reasonable for a community detection algorithm only to divide up networks when good divisions exist and to leave them undivided the rest of the time.

Why partitioning is hard?

- Example: graph *bisection*
- Formally the graph bisection problem is the problem of dividing the vertices of a network into two 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*
- One might imagine that one could bisect a network simply by looking through all possible divisions of the network into two parts of the required sizes and choosing the one with the smallest cut size.
- n_1 and n_2 are the group sizes: $n_1 + n_2 = n$
- The number of ways of how we can make the division $\frac{n(n-1)\ldots(n-n_1-1)}{n_1!} = \frac{n!}{n_1!n_2!}$
- Stirling's formula: $n! \approx \sqrt{2\pi n}(n/e)^n$
- $\frac{n!}{n_1!n_2!} \approx \frac{n^{n+1/2}}{n_1^{n_1+1/2}n_2^{n_2+1/2}}$
- If $n_1 = n_2 = \frac{n}{2}$ this number is $\frac{2^{n+1}}{\sqrt{n}}$
- So the amount of time required to look through all of these divisions will go up roughly exponentially with the size of the network.
- **The task is not to find the best partition, but the "pretty good" one**

The Kernighan-Lin algorithm

- Proposed by Brian Kernighan and Shen Lin in 1970. The simplest and best known heuristic algorithms for the graph bisection problem.
- We start by dividing the vertices of our network into two groups of the required sizes in any way we like.
- Then, for each pair (i, j) of vertices such that i lies in one of the groups and j in the other, we calculate how much the cut size between the groups would change if we were to interchange i and j
- Among all pairs (i, j) we find the pair that reduces the cut size by the largest amount or, if no pair reduces it, we find the pair that increases it by the smallest amount.
- We swap that pair of vertices.
- The process is then repeated, but with the important restriction that each vertex in the network can only be moved once.
- And so the algorithm proceeds, swapping on each step that pair that most decreases, or least increases, the number of edges between our two groups, until eventually there are no pairs left to be swapped, at which point we stop. (If the sizes of the groups are unequal then there will be vertices in the larger group that never get swapped, equal in number to the difference between the sizes of the groups.)

The Kernighan-Lin algorithm

- When all swaps have been completed, we 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.
- Finally, this entire process is performed repeatedly, starting each time with the best division of the network found on the last time around and continuing until no improvement in the cut size occurs. The division with the best cut size on the last round is the final division returned by the algorithm.
- Once we can divide a network into two pieces of given size then we can divide it into more than two simply by repeating the process. For instance, if we want to divide a network into three pieces of equal size, we would first divide into two pieces, one twice the size of the other, and then further divide the larger one into two equally sized halves.
- **Important note:** two different random starting states could (though needn't necessarily) result in different divisions of the network.
- **Disadvantage:** the algorithm is quite slow. One needs $O(mn^2)$ operations, where m is the number of edges.

Spectral partitioning

- We describe the spectral partitioning method as applied to the graph bisection problem, the problem of dividing a graph into two parts of specified sizes.
- Division into more than two groups is typically achieved by repeated bisection, dividing and subdividing the network to give groups of the desired number and size.
- Consider a network of n vertices and m edges and a division of that network into two groups, which we will call group 1 and group 2.
- Let R be the cut size for the division

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

- Let us define the set of quantities $s_i,\ i = 1, \ldots, n$

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

Spectral partitioning

- $\frac{1}{2}(1 - s_i s_j) = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are in different groups} \\ 0, & \text{otherwise} \end{cases}$
- The cut size, therefore, can be calculated as follows

$$R = \frac{1}{4} \sum_{ij} A_{ij}(1 - s_i s_j) = \frac{1}{4} \Big(\sum_{ij} A_{ij} - \sum_{ij} A_{ij} s_i s_j \Big)$$

- $\sum_{ij} A_{ij} = \sum_i d_i = \sum_i d_i s_i^2 = \sum_{ij} k_i \delta_{ij} s_i s_j$
- $R = \frac{1}{4} \sum_{ij} (d_i \delta_{ij} - A_{ij}) s_i s_j \equiv \frac{1}{4} \sum_{ij} L_{ij} s_i s_j$,
  where $\mathbf{L} = \mathbf{D} - \mathbf{A}$, $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ is the graph Laplacian matrix.

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

- **Partition problem:** find the vector $\mathbf{s}$ that minimizes the cut size R for given $\mathbf{L}$.

Spectral partitioning

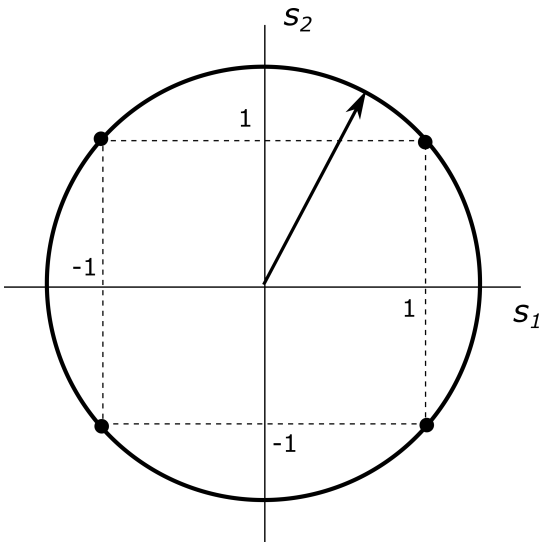
- **Partition problem:** find the vector $\mathbf{s}$ that minimizes the cut size R for given $\mathbf{L}$.
- This discrete valued optimization problem is as complex as partition problem in general.
- What makes our matrix version of the problem hard in practice is that the s_i are discrete: ± 1
- If they were allowed to take any real values the problem would be much easier. We could just differentiate to find the optimum.
- This suggests a possible approximate approach to the minimization problem.
  - Suppose s_i can take any values from some range. We can then find the values that minimize R.
  - These values will only be approximately the correct ones, since they probably won't be ± 1, but they may nonetheless be **good enough** to give us a approximation of the optimal partitioning.
  - *Relaxation method*

Spectral partitioning

- In the exact problem $s_i = \pm 1,\ i = 1, \ldots, n$

$$\sum_i s_i^2 = n \tag{1}$$

- If we regard s as a vector of the Euclidean 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, and always has the same length, which is $\sqrt{n}$.
- Relaxation: 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. Or, equivalently, the constraint (1) holds.



Spectral partitioning

- The second constraint on $s_i,\ i = 1, \ldots, n$ is

$$\sum_i s_i = n_1 - n_2, \tag{2}$$

  where n_1 (n_2) is the size of the group 1 (2). In vector notation:

$$\mathbf{1}^T \mathbf{s} = n_1 - n_2$$

- We keep this second constraint unchanged in our relaxed calculations
- Our partitioning problem, **in its relaxed form**, is a problem of minimizing the cut size subject to the two constraints
- This problem is now just a standard piece of algebra.

Spectral partitioning

- Lagrangian $\mathcal{L}$ is written as follows

$$\mathcal{L} = \sum_{jk} L_{jk} s_j s_k + \lambda(n - \sum_j s_j^2) + 2\mu((n_1 - n_2) - \sum_j s_j),$$

where λ and 2μ are two Lagrange multipliers.

- We differentiate $\mathcal{L}$ with respect to the elements s_i:

$$\frac{\partial}{\partial s_i}\mathcal{L} = 0$$

Which leads to the following equations

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

or, in matrix notation

$$\mathbf{Ls} = \lambda \mathbf{s} + \mu \mathbf{1} \tag{3}$$

Spectral partitioning

- Note that $\mathbf{L1} = 0$.
- Multiplying (3) on the left by $\mathbf{1}^T$ we obtain

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

- Let us introduce vector $\mathbf{x}$ as follows

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

- Note that

$$\mathbf{Lx} = \mathbf{L}\left(\mathbf{s} + \frac{\mu}{\lambda}\mathbf{1}\right) = \mathbf{Ls} = \lambda\mathbf{s} + \mu\mathbf{1} = \lambda\mathbf{x}$$

  $\mathbf{x}$ is an eigenvector of the Laplacian with eigenvalue λ.

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

  $\mathbf{x}$ is orthogonal to $(1, 1, \ldots, 1)$. Thus, it cannot be the eigenvector $(1, 1, 1, ...)$ that has eigenvalue zero.

Spectral partitioning

- Note that

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

-

$$\begin{aligned}\mathbf{x}^T\mathbf{x} &= \mathbf{s}^T\mathbf{s} + \frac{\mu}{\lambda}\left(\mathbf{s}^T\mathbf{1} + \mathbf{1}^T\mathbf{s}\right) + \frac{\mu^2}{\lambda^2}\mathbf{1}^T\mathbf{1} \\ &= n - 2\frac{n_1 - n_2}{n}(n_1 - n_2) + \frac{(n_1 - n_2)^2}{n} \\ &= \frac{(n_1 + n_2)^2 - (n_1 - n_2)^2}{n} = 4\frac{n_1 n_2}{n}\end{aligned} \tag{4}$$

  Therefore,

$$R = \frac{n_1 n_2}{n}\lambda \tag{5}$$

- Thus, the cut size is proportional to the eigenvalue λ. Given that our goal is to minimize R, this means we should choose $\mathbf{x}$ to be the eigenvector corresponding to the smallest allowed eigenvalue of the Laplacian.

Spectral partitioning

- All the eigenvalues of the Laplacian are non-negative (Laplacian property).
- The smallest one is the zero eigenvalue that corresponds to the eigenvector $(1, 1, 1, \dots)$ but we have already ruled this one out, as $\mathbf{x}$ has to be orthogonal to this lowest eigenvector.
- Thus, the best thing we can do is choose $\mathbf{x}$ proportional to the eigenvector $\mathbf{v}_2$ corresponding to the second lowest eigenvalue λ_2 with its normalization fixed by (4).
- Finally, we recover the corresponding value of **relaxed** $\mathbf{s}$

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

- In the solution of the initial problem we should have $s_i = \pm 1,\ i = 1, \dots, n$. Moreover, we should have exactly n_1 values with $+1$ and n_2 – with -1.

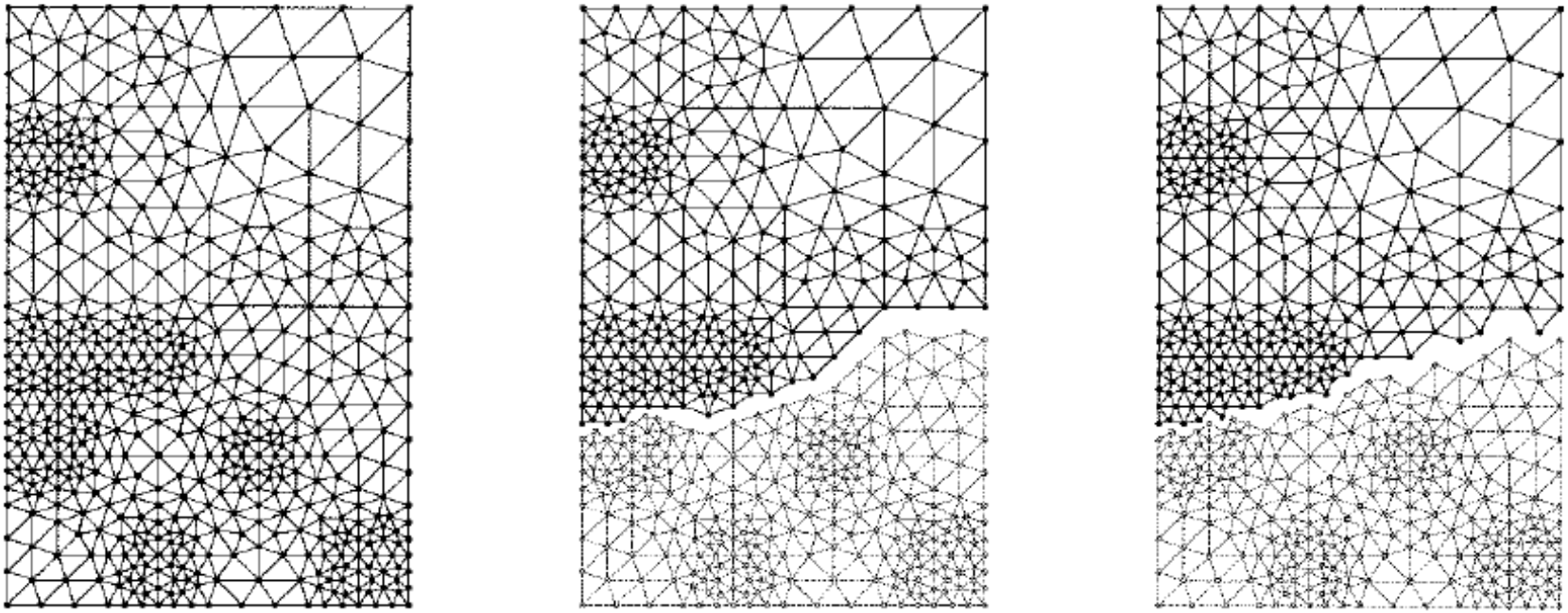
Spectral partitioning

- Let us do the best we can and choose $\mathbf{s}$ to be as close as possible to our ideal relaxed value subject to its constraints.
- We do this by making the product

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

  as large as possible.
- To do this we should put $s_i = +1$ for the nodes corresponding to the n_1 maximal $(x_i + \frac{n_1 - n_2}{n})$ values $\longleftrightarrow$ n_1 maximal x_i values. For the others we put $s_i = -1$.
- The speed of the spectral method is $O(mn)$ which is much better than in the Kernighan-Lin algorithm

Example



(a) (b) (c)

Figure 11.3: Graph partitioning applied to a small mesh network. (a) A mesh network of 547 vertices of the kind commonly used in finite element analysis. (b) The edges removed indicate the best division of the network into parts of 273 and 274 vertices found by the Kernighan–Lin algorithm. (c) The best division found by spectral partitioning. The network is from Bern *et al.* [35].