V4 Matrix algorithms and graph partitioning

- Community detection
- Simple modularity maximization
- Spectral modularity maximization
- Division into more than two groups
- Other algorithms for community detection

Community detection

Basic **goal** of **community detection**:

We want to separate the network into groups of vertices
that have **few connections between** them.

The important difference to graph partitioning is that
the number or size of the groups is **not fixed** anymore.

Simplest task: divide graph into 2 groups or communities
but without any constraint on the size of the groups.

First idea: choose division with **minimum cut size**.

BUT this will not work. An optimal solution would be to simply put all vertices into one group and none into the other one. Then the cut size R is zero.

Community detection

Another way would be to impose some loose constraint on the sizes of the groups n_1 and n_2.

One example of this kind is **ratio cut partitioning** where one minimizes the ratio $\frac{R}{n_1 n_2}$.

The denominator $n_1 \times n_2$ has its largest value when both groups are equal size.

$1 \times 9 = 9$
$2 \times 8 = 16$
$3 \times 7 = 21$
$4 \times 6 = 24$
$5 \times 5 = 25$
$6 \times 4 = 24$ …

This biases the optimization always to solutions where the groups have roughly equal size.

However, there is not particular reason behind such an optimization principle.

Community detection

What different measures could be used to quantify the quality of a division besides the simple cut size or its variants?

A good division is one where there are **fewer than expected** edges between groups.

→ apply the **modularity** measure used for **assortative mixing** (see V2).

Review (V2): Quantify assortative mixing

Find the fraction of edges that run between vertices of the same type and subtract from this the fraction of edges we would expect if edges were positioned at random without considering the vertex type.

c_i : class or type of vertex i , $c_i \in [1 \ldots n_c]$
n_c : total number of classes

The total number of edges between vertices of the same type is

$$\sum_{\text{edges } (i,j)} \delta(c_i, c_j) = \frac{1}{2} \sum_{ij} A_{ij} \delta(c_i, c_j)$$

Here $\delta(m,n)$ is the Kronecker delta (δ is 1 if $m = n$ and 0 otherwise). The factor ½ accounts for the fact that every vertex pair i,j is counted twice in the sum.

(Review V2): Quantify assortative mixing

As expected number of edges between all pairs of vertices of the same type we derived

$$\ldots\ldots\ldots \quad \frac{1}{2}\sum_{ij}\frac{k_i k_j}{2m}\delta(c_i, c_j)$$

where the factor ½ avoids double-counting vertex pairs.

Taking the difference between the actual and expected number of edges gives

$$\frac{1}{2}\sum_{ij} A_{ij}\delta(c_i, c_j) - \frac{1}{2}\sum_{ij}\frac{k_i k_j}{2m}\delta(c_i, c_j) = \frac{1}{2}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j)$$

Typically one does not calculate the number of such edges but the fraction, which is obtained by dividing this by *m*

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j)$$

This quantity Q is called the **modularity**.

Modularity maximization by Kernighan-Lin algorithm

We will design an analog of the Kernighan-Lin algorithm where we are not required to swap pairs of vertices at every step. Instead we now swap single vertices.

- At each step, we select the vertex whose movement would most increase, or least decrease, the modularity. In a full cycle, each vertex is moved exactly once.

- Then go back over the states through which the network has passed and select the one with the highest modularity.

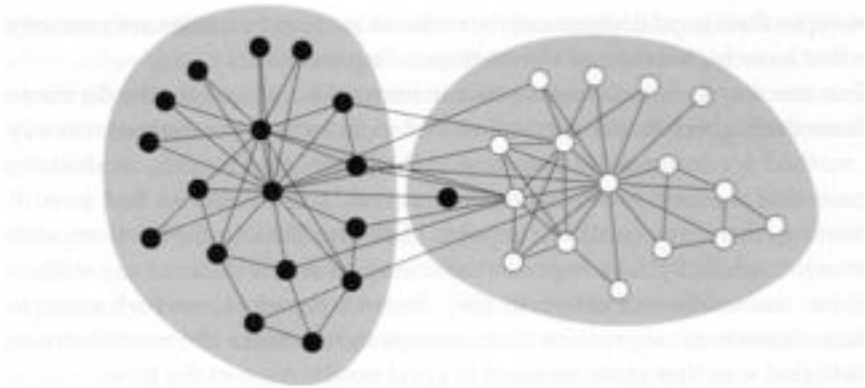
- Use this state as starting condition for another round of the algorithm.

- Repeat this until the modularity no longer improves.

The complexity is now lower, $O(mn)$, because when moving single vertices we only have to consider $O(m)$ possible moves at each step, in contrast to $O(m^2)$ pairs.

Example: „karate club“ network of Zachary

Example for application of Kernighan-Lin modularity maximization.



Pattern of friendship between 34 members of a karate club at a US university.

At a later point in time, a dispute arose between the members of the club whether the club‘s fees should be raised.
This led to a splitting up of the club into two parts with 16 and 18 members each.
The „true“ groups after the division are colored black and white in the figuure.

The communities detected by modularity maximization
correspond almost perfectly to the formed groups.

Spectral modularity maximization

There is also an analog of the spectral graph partitioning algorithm for community detection.

The modularity of a division can be rewritten into:

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j) = \frac{1}{2m}\sum_{ij} B_{ij}\delta\left(c_i, c_j\right)$$

$$\text{with } B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

where B_{ij} has the property

$$\sum_j B_{ij} = \sum_j A_{ij} - \frac{k_i}{2m}\sum_j k_j = k_i - \frac{k_i}{2m}2m = 0$$

Since every edge in an undirected graph has 2 ends, *m* edges have 2*m* ends.

Spectral modularity maximization

We will start again by dividing the network into 2 parts and define

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

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

Substituting this into the previous equation gives

$$Q = \frac{1}{2m}\sum_{ij} B_{ij}\delta(c_i, c_j) = \frac{1}{4m}\sum_{ij} B_{ij}(s_i s_j + 1) = \frac{1}{4m}\sum_{ij} B_{ij} s_i s_j$$

where we have used $\sum_j B_{ij} = 0$

In matrix notation, we can write this as $Q = \frac{1}{4m}\mathbf{s}^T \mathbf{B}\, \mathbf{s}$

B is also called the **modularity matrix**.

Spectral modularity maximization

This equation is similar to the former expression in V3
for the cut size of a network in terms of the graph Laplacian.

→ we can derive a spectral algorithm for community detection
that is closely analogous to the spectral partioning method.

We wish to find a division **s** of the network that maximizes
the modularity Q for a give modularity matrix **B**.

The elements of **s** are constrained to take values of +1 and -1.

Contrary to before, the number of elements with +1 and -1 values is now not fixed.

As before, we will relax the constraint that **s** must point into
the corners of a hypercube and allow it to point in any direction
subject to the constraint that its length is kept the same: $\mathbf{s}^T\mathbf{s} = \sum_i {s_i}^2 = n$

Spectral modularity maximization

To find the maximum of $Q = \frac{1}{4m}\mathbf{s}^T\mathbf{B}\,\mathbf{s}$ we differentiate and impose the constraint with a single Lagrangian multiplier β

$$\frac{\partial}{\partial s_i}\left[\sum_{jk} B_{jk}s_j s_k + \beta\left(n - \sum_j {s_j}^2\right)\right] = 0$$

This gives $2\sum_j B_{ij}s_j - 2\beta s_i = 0$ or in matrix notation **B s** = β **s**

In other words, **s** is one of the eigenvectors of the modularity matrix.

With this solution, the modularity then becomes

$$Q = \frac{1}{4m}\mathbf{s}^T\mathbf{B}\,\mathbf{s} = \frac{1}{4m}\beta\mathbf{s}^T\mathbf{s} = \frac{n}{4m}\beta$$

For maximum modularity, we should choose **s** to be the eigenvector $\mathbf{u}_1$ corresponding to the largest eigenvalue of the modularity matrix.

Spectral modularity maximization

But as before, we typically cannot choose $\mathbf{s} = \mathbf{u}_1$
since the elements of **s** are subject to the constraint $s_i = \pm 1$.

Instead we will do the best we can and choose **s** as close to $\mathbf{u}_1$
as possible by maximizing their scalar product:

$$\mathbf{s}^T \mathbf{u}_1 = \sum_i s_i [\mathbf{u}_1]_i$$

where $[\mathbf{u}_1]_i$ is the *i*-th element of $\mathbf{u}_1$.

The maximum is achieved when each term in the sum is non-negative, i.e. when

$$s_i = \begin{cases} +1 & \text{if } [\mathbf{u}_1]_i > 0 \\ -1 & \text{if } [\mathbf{u}_1]_i < 0 \end{cases}$$

If a vector element is exactly zero, either value of s_i is equally good.

Spectral modularity maximization

This yields the following simple algorithm:

- Calculate the eigenvector of the modularity matrix corresponding to the largest (most positive) eigenvalue
- Then assign vertices to communities according to the signs of the vector elements, positive signs in one group and negative signs in the other.

In practice, this works very well. E.g. the karate club is perfectly classified.

In practical applications, it is worthwhile to use spectral modularity maximization as a first step, followed by the Kernighan-Lin method to get some small further improvements.

Division into more than 2 groups

In general, networks have an **arbitrary number** of communities.

Modularity is supposed to be largest for the best division of the network.

As first method, we could start by dividing the network into 2 parts,
and then further subdivide those parts into smaller ones, and so forth.

However, the modularity of the complete network does not break up (as the cut size does) into independent contributions from the separate communities.

The individual maximization of the modularities of these communities treated as separate networks will generally not produce the maximum modularity for the network as a whole.

Division into more than 2 groups

We must consider explicitly the change ΔQ in the modularity of the entire network upon further bisecting a community *c* of size n_c.

By comparing the modularity after and before the division, we find (without derivation)

$$\Delta Q = \cdots = \frac{1}{4m}\mathbf{s}^T\mathbf{B}^{(c)}\mathbf{s}$$

$\mathbf{B}^{(c)}$ is the $n_c \times n_c$ matrix with elements

$$B_{ij}{}^{(c)} = B_{ij} - \delta_{ij}\sum_{k\in c} B_{ik}$$

Division into more than 2 groups

The equation $\Delta Q = \frac{1}{4m}\mathbf{s}^T\mathbf{B}^{(c)}\mathbf{s}$ has the same form as before.

Thus, we can apply our spectral approach to this generalized modularity matrix to maximize ΔQ, find the leading eigenvector and divide the network according to the signs of its elements.

In repeatedly subdividing a network in this way, an important question is when to stop.

The answer is simply when we are unable to find a (further) division with a positive change ΔQ in the modularity.

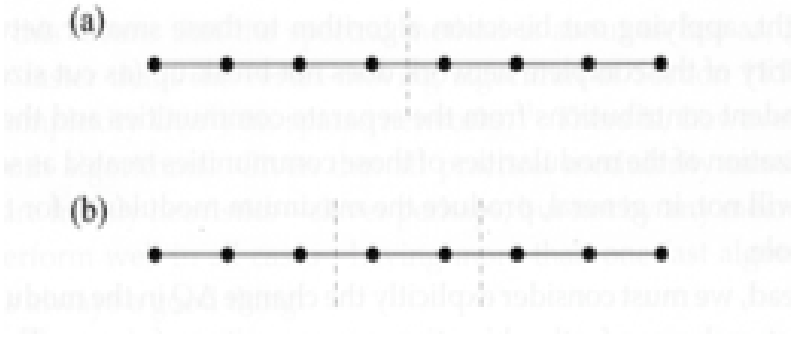
In that case, the bisection algorithm will put all vertices in one of its 2 groups and none in the other, effectively „refusing“ to further subdivide the network.

At this point we should stop.

Division into more than 2 groups

The repeated bisection method works well in many situations, but it is by no means perfect.

A particular problem is that there is no guarantee that the best division of a network in, say 3 parts, can be found by first finding the best division into 2 parts and then subdividing one of them.



(a) shows the best subdivision of this linear graph with 8 vertices and 7 edges into two groups with 4 vertices each.

(b) Shows the best subdivision into 3 groups with 3, 2, 3 vertices each.

A repeated bisection algorithm would never find solution (b).

Other algorithms for community detection

An alternative way of finding communities of vertices in a network is to loook for the edges that lie between communities.

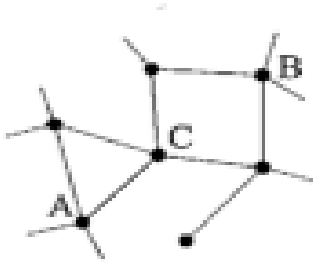
If we can find and remove these edges, we will be left with isolated communities.

One common way to define „betweenness“ is to use **betweenness centrality** (V1).

Review (V2): Betweenness Centrality

Let us assume an undirected network for simplicity.

Let n_{st}^{i} be 1 if vertex *i* lies on the geodesic path from *s* to *t*
and 0 if it does not or if there is no such path.



Vertices A and B are connected by 2 geodesic paths. Vertex C lies on both paths.

Then the **betweenness centrality** x_i is given by

$$x_i = \sum_{st} n_{st}^{i}$$

This definition counts separately the geodesic paths in either direction between each vertex pair.

The equation also includes paths from each vertex to itself.
Excluding those would not change the ranking of the vertices in terms of betweenness.

Also, we assume that vertices *s* and *t* belong to paths between *s* and *t.*

If there are two geodesic paths of the same length between 2 vertices,
each path gets a weight equal to the inverse of the number of paths.

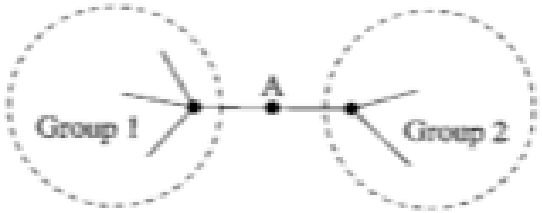
Review (V2): Betweenness Centrality

We may redefine n_{st}^{i} to be the number of geodesic paths from *s* to *t* that pass through vertex *i,* and define g_{st} to be the total number of geodesic paths from *s* to *t*.

Then, the betweenness centrality of x_i is

$$x_i = \sum_{st} \frac{n_{st}{}^{i}}{g_{st}}$$

where we adopt the convention that $n_{st}^{i} / g_{st} = 0$ if both n_{st}^{i} and g_{st} are zero.



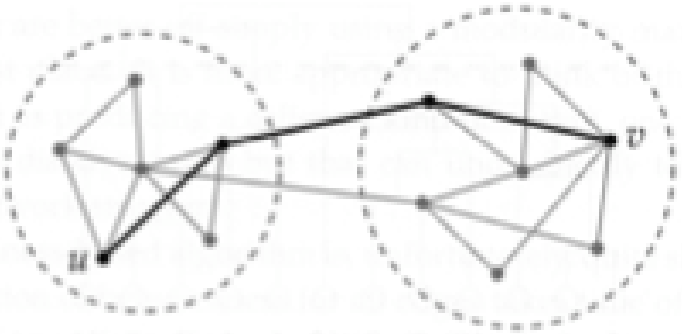
In this sketch of a network, vertex A lies on a bridge joining two groups of other vertices.
All paths between the groups must pass through A, so it has a high betweenness even though its degree is low.

Use edge betweenness for community detection

Define **edge betweenness** as the number of geodesic paths that run along particular edges.

We expect that edges that lie between communities will have high values of this edge betweenness.

In this example, two edges run between the vertices in the two dashed circles. All shortest paths between vertices of the two groups will run along one of these two edges.



Edge betweenness is computed by determining the geodesic paths between every pair of vertices in the network and count how many such paths go along each edge. This takes $O(n(m + n))$.

Betweenness algorithm

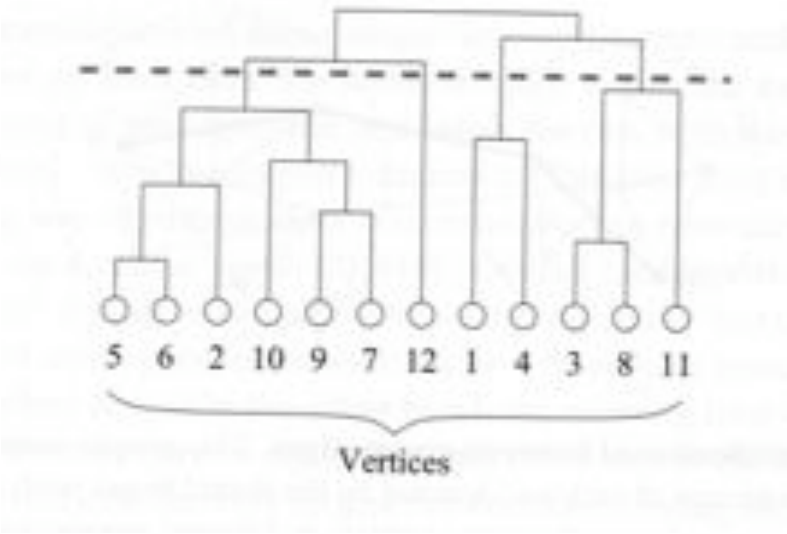
1. Calculate betweeness score of all edges
2. Find the edge with the highest score and remove it.

Because removing this edge will change the betweenness scores of some edges, any shortest paths that previously traversed the removed edge will now have to be rerouted. Thus we have to go back to step 1.

The progress of the algorithm can be represented using a **tree**.

The progressive fragmentation of the network as edged are removed one by one is represented by the successive branching of the tree.

If we stop at the dashed line, the network is split into 4 groups of 6, 1, 2, and 3 vertices.



Hierarchical clustering

Network division by edge betweenness produced a dendrogram that is remiscent of clustering methods.

Hierarchical clustering is an entire class of algorithms.
It is an **agglomerative** technique where we start with the individual vertices of a network and join them together to form groups.

We need a measure of vertex similarity. For this, we can use the measures introduced in V1 and V2, e.g.

- cosine similarity,
- correlation coefficients between rows of the adjacency matrix,
- or the so-called Euclidian distance.

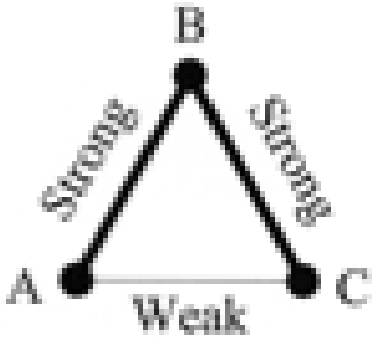
Having many choices for similarity measures is both a strength and a weakness of hierarchical clustering methods.

It gives the method flexibility, but the results will differ from one another.

Hierarchical clustering

Once a similarity is chosen we calculate the similarity of all pairs of vertices in the network.

Then we would like to connect the most similar vertices.



However, there may be conflicting situations.
Should A and C be in the same group or not?

The basic strategy of hierarchical clustering is to start by joining those pairs of vertices with the highest similarities. These then form groups of size 2.
This step involves no ambiguity.

Then we further join together the groups that are most similar to form larger groups, and so on.

Hierarchical clustering

During this process we now require a measure for the similarity of groups.

There are 3 common ways of combining vertex similarities to give similarity scores for groups:
- single-linkage
- complete-linkage
- average-linkage clustering.

Consider 2 groups of vertices, group 1 and group 2, with n_1 and n_2 vertices, respectively.

Then there are $n_1 n_2$ pairs of vertices such that one vertex is in group 1 and the other in group 2.

In the **single-linkage clustering** method, the similarity between the 2 groups is defined as the similarity of the most similar of these $n_1 n_2$ pairs of vertices.

Hierarchical clustering

In the **single-linkage clustering** method, the similarity between the 2 groups is defined as the similarity of the most similar of these $n_1 n_2$ pairs of vertices.

As the other extreme, **complete-linkage clustering** defines the similarity between the 2 groups as the similarity of the **least similar** pair of vertices.

In between these two extremes is the **average-linkage clustering** where the similarity of two groups is defined to be the mean similarity of all pairs of vertices.

Hierarchical clustering

This is how hierarchical clustering is done:

1. Choose a similarity measure and evaluate it for all vertex pairs.
2. Assign each vertex to a group of its own, consisting of just that one vertex. The initial similarities of the groups are simply the similarities of the vertices

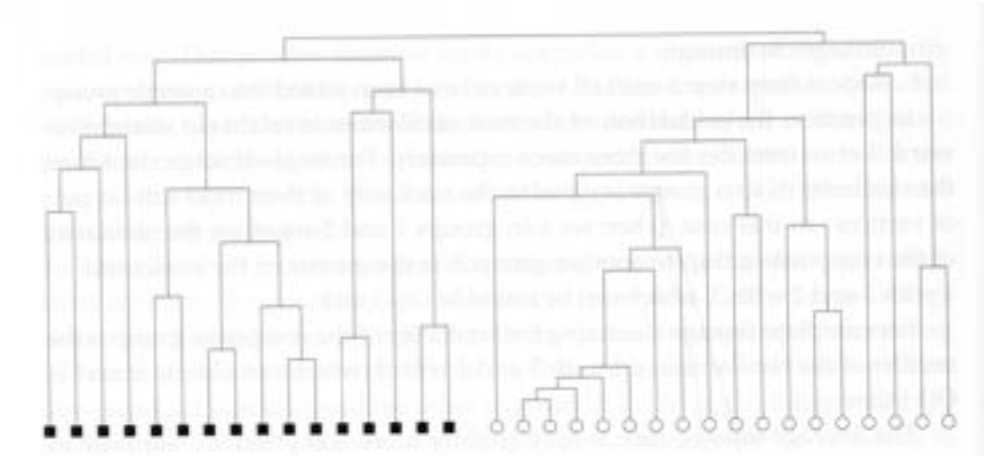
3. Find the pair of groups with the highest similarity and join them together into a single group.

4. Calculate the similarity between the new composite group and all others using one of the 3 methods above (single, complete, average-linkage)

5. Repeat from step 3 until all vertices have been joined into a single group.

Hierarchical clustering applied to the karate club

Partitioning of the karate club network by average linkage hierarchical clustering using cosine similarity as our measure of vertex similarity.



For this example, hierarchical clustering found the perfect division of the club.

However, hierarchical clustering does not always work as well as here.

Comparison of modularity maximization methods

A large number of approaches have been developed to maximize modularity for divisions into any number of communities of any sizes.

Author	Ref.	Label	Order
Eckmann & Moses	[13]	EM	$O(m\langle k^2\rangle)$
Zhou & Lipowsky	[14]	ZL	$O(n^3)$
Latapy & Pons	[15]	LP	$O(n^3)$
Newman	[24]	NF	$O(n\log^2 n)$
Newman & Girvan	[25]	NG	$O(m^2 n)$
Girvan & Newman	[32]	GN	$O(n^2 m)$
Guimerà et al.	[27, 43]	SA	parameter dependent
Duch & Arenas	[31]	DA	$O(n^2 \log n)$
Fortunato et al.	[33]	FLM	$O(n^4)$
Radicchi et al.	[34]	RCCLP	$O(n^2)$
Donetti & Muñoz	[35, 36]	DM/DMN	$O(n^3)$
Bagrow & Bollt	[37]	BB	$O(n^3)$
Capocci et al.	[38]	CSCC	$O(n^2)$
Wu & Huberman	[39]	WH	$O(n+m)$
Palla et al.	[40]	PK	$O(\exp(n))$
Reichardt & Bornholdt	[41]	RB	parameter dependent

Table 1. Table summarising how the computational cost of different approaches scales with number of nodes n, number of links m and average degree $\langle k\rangle$ [42]. The labels shown here are used in Figures 2 and 3.

Danon, Duch, Diaz-Guilera, Arenas, J. Stat. Mech. P09008 (2005)

Comparison of modularity maximization methods

One way to test the **sensitivity** of these methods is to see how well a particular method performs when applied to ad hoc networks with a well known, fixed community structure. Such networks are typically generated with n = 128 nodes, split into 4 communities containing 32 nodes each.

Pairs of nodes belonging to the same community are linked with probability p_{in} whereas pairs belonging to different communities are joined with probability p_{out}. The value of p_{out} is taken so that the average number of links a node has to members of any other community, z_{out}, can be controlled.

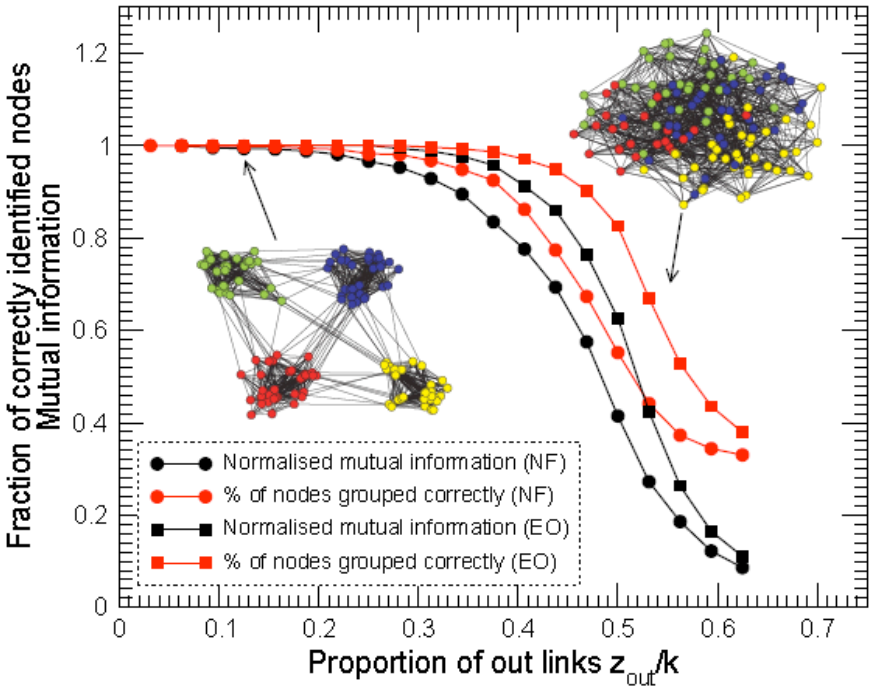
While p_{out} (and therefore z_{out}) is varied freely, the value of p_{in} is chosen to keep the total average node degree, k constant, and set to 16.

Danon, Duch, Diaz-Guilera, Arenas, J. Stat. Mech. P09008 (2005)

Comparison of modularity maximization methods

As z_{out} increases, the communities become more and more diffuse and harder to identify, (see figure).

Since the “real” community structure is well known in this case, it is possible to measure the number of nodes correctly classified by the method of community identification.



Danon, Duch, Diaz-Guilera, Arenas, J. Stat. Mech. P09008 (2005)

Other modularity maximization methods

One of the most successful approaches is **simulated annealing**.

The process begins with any initial partition of the nodes into communities.

At each step, a node is chosen at random and moved to a different community, also chosen at random.

If the change improves the modularity it is always accepted, otherwise it is accepted with a probability $\exp(\Delta Q/kT)$.

The simulation will start at high temperature T and is then slowly cooled down.

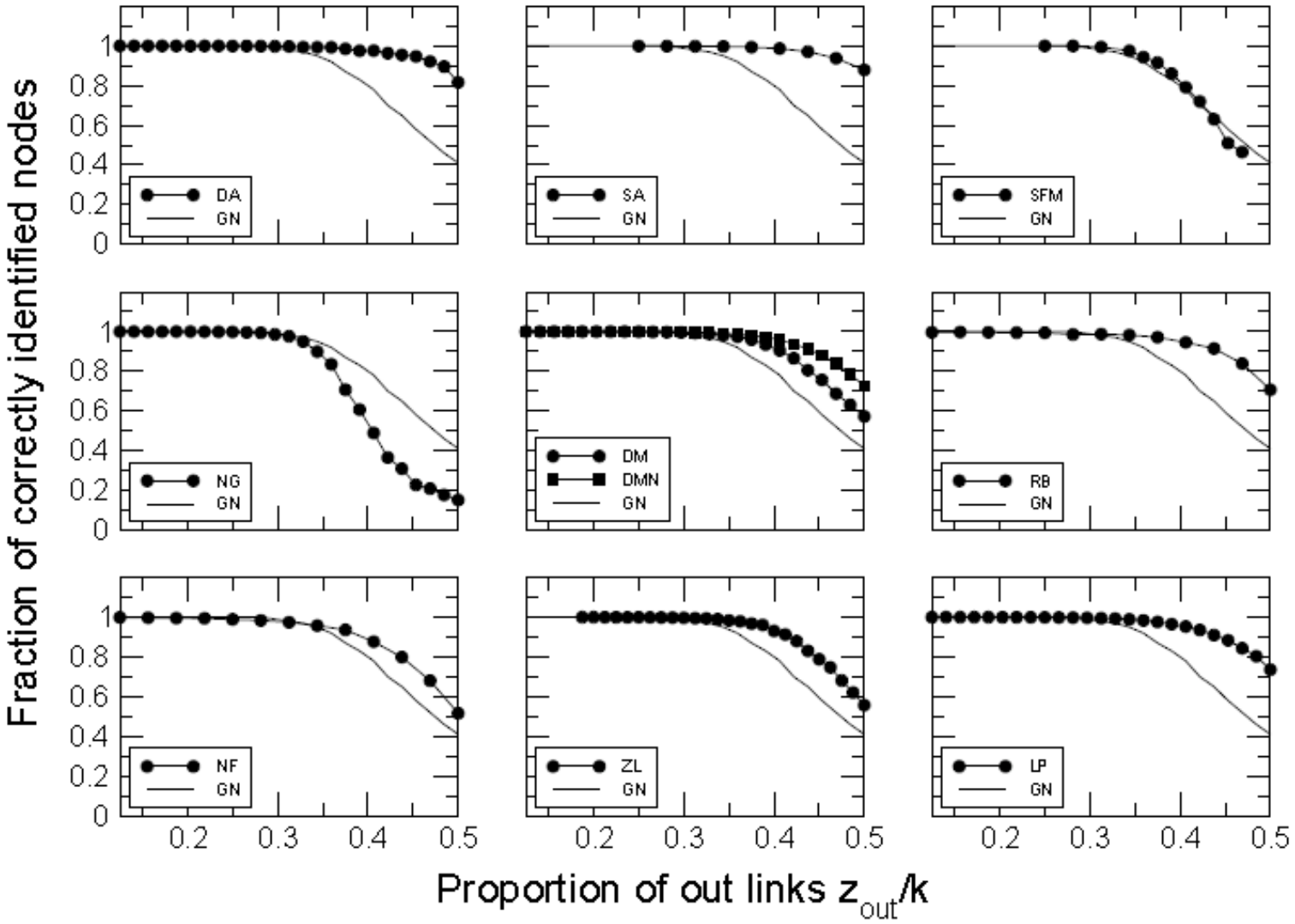
Several improvements have been tested.

Firstly, the algorithm is stopped periodically, or quenched,

and ΔQ is calculated for moving each node to every community that is not its own.

Finally, the move corresponding to the largest value of ΔQ is accepted.

Comparison of modularity maximization methods



Danon, Duch, Diaz-Guilera, Arenas, J. Stat. Mech. P09008 (2005)