Pairwise Node Similarity Measures for Role Detection in Networks

Candidate Number 1024173[a]

[a]Mathematical Institute, University of Oxford, Andrew Wiles Building, Radcliffe Observatory Quarter, Woodstock Road, Oxford OX2 6GG

We describe and implement two methods based on pairwise node similarity for detecting block structures in networks. We analyse their performance and show them to effectively uncover particular block structures. We discuss their respective shortcomings and briefly describe an improvement to one of the methods.

Role Detection | Networks | Pairwise Node Similarity

Community detection (1) is one of the most fundamental problems in the analysis of complex networks and has received much attention over the past 20 years. This is in part due to its wide reaching applications in the fields of technology, social science and biology. A *community* in a network tends not to be given a formal mathematical definition, but is described as a group of nodes which are densely connected to each other, and sparsely connected to other nodes in the network. For example in a social network, communities might represent close-knit circles of friends who interact frequently with each other, and less frequently with others.

This report is focused on the problem of *role detection*. Given a network, can we determine which nodes have similar *roles* or *functions*? For example in a biological food web, we might view organisms in the same trophic level as having similar roles (2). Or in a network of the sales of goods, roles might be producers, retailers and consumers (3). This is a more general problem than finding communities. There is no reason that in general, nodes playing similar roles in the network should be densely connected to one another. We will later consider how we might mathematically describe what it means for nodes to have similar roles. (Note that community detection is a specific example of role detection, where the chosen definition of having the same role is being densely connected to one another.)

The rest of the report is organised as follows. We introduce the *Stochastic Block Model* (SBM) and consider first examples of what it might mean for nodes to have similar roles. We then mention two possible strategies for role detection before more fully describing two other methods (2) and (4) which are based on the notion of *pairwise node similarity*. We implement these two algorithms in Python and examine their effectiveness. We also compare their performance when applied to a common network and infer which types of role structure each method is more suited to detecting. We then highlight the drawbacks of the two methods and briefly describe an extension to one of them (5). Finally, we offer a discussion on modelling issues and empirical data.

Stochastic Block Model. When organised according to its community or role structure, the adjacency matrix of a network takes a block form, see Fig. 1. Finding such latent block structures in networks is known as *block modelling* or *role detection*. To test the effectiveness of our methods, we would like a way of generating networks with block structure, so that

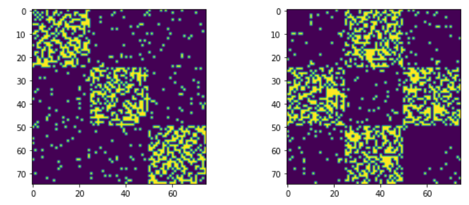

Fig. 1. Adjacency matrices of two networks with block structure.

we can see how well the algorithms perform at detecting them. Here we describe the *stochastic block model*. The stochastic block model is a random graph model, taking the following parameters:

- a positive integer n,
- a partition of $\{1, 2, \ldots, n\}$ into disjoint subsets $C_1, C_2, \ldots, C_r$,
- a symmetric $r \times r$ matrix $\mathbf{P} = (p_{ij})$.

The integer n is the number of vertices and $C_1, C_2, \ldots, C_r$ describes the splitting of the n vertices into r blocks. An edge between a node in the ith block and a node in the jth block is present (independently of the other edges) with probability p_{ij}.

We could use the stochastic block model to produce a random graph with a community structure by taking the edge probabilities within blocks p_{ii} to be relatively larger, and the probabilities between blocks p_{ij} (for $i \neq j$) to be relatively smaller. Other choices for $\mathbf{P}$ lead to other block structures.

Role Modelling

Communities in networks are characterised by clusters of nodes being densely connected to each other. While we might then interpret this on an empirical network as representing

Significance Statement

While community detection has been widely studied, the more general problem of role detection has been given less attention. Discovering the function which nodes play in a network could have significant implications. For example, greater knowledge of how information is transferred and flows through a network of individuals could lead to more informed advertising strategies. Meanwhile a better understanding of roles could improve detection of anomalies in IP networks (6). We describe two simple role detection methods and provide implementations of them in Python.

for example, close-knit groups of friends, the description of community is independent of the interpretation; it is given purely in terms of the underlying graph structure. We would like to come up with a similar mathematical description of role classes. We will consider various formulations when we consider role detection algorithms, but as a first example we start with the notions of *structural* and *regular equivalence.*

Two nodes in a network are said to be *structurally equivalent* if they have exactly the same neighbours (3). This is a very strict criterion. It is very possible that every node simply ends up in a structural equivalence class of its own (Fig. 2).

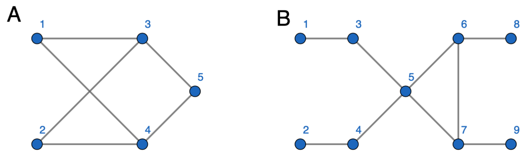

Fig. 2. In network A, nodes 1 and 2 have exactly the same neighbors and so are structurally equivalent, as are 3 and 4. In network B, no two nodes are structurally equivalent.

A slight relaxation leads to the notion of *regular equivalence.* (The following description is based on that in (3).) In loose terms, two nodes are *regularly equivalent* if they are connected in the same way to equivalent others. For each regular equivalence class, we have one node in a new *image* or *reduced* graph. Given two classes A and B, if there exists an edge in the original graph with one end in A and one end in B, then we put an edge between A and B in the image graph.

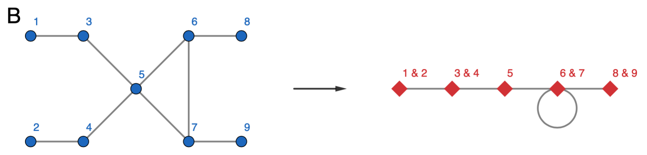

Fig. 3. The pairs of nodes (1,2), (3,4), (6,7) and (8,9) each form a class of regular equivalence

An example of role detection would therefore be trying to find these structural or regular equivalence classes.

Alternative approaches

Optimisation. A possible direction for investigating role detection is Reichardt and White's *optimisation* algorithm (3). The concept is similar to that of modularity optimisation; in fact it is a generalisation of it.

As with modularity, they define a *quality function* $\mathcal{Q}^B$. Given an image graph and an assignment of roles to the network, $\mathcal{Q}^B$ provides a measure of how well the assignment fits the image graph. A large value of $\mathcal{Q}^B$ corresponds to the image graph being a good descriptor of the connection structure of the network.

Given a network with M edges and adjacency matrix $\mathbf{A} = (A_{ij})$, and an image graph with q roles and adjacency matrix $\mathbf{B} = (B_{rs})$, consider an assignment of roles $\sigma_i \in \{1, \ldots, q\}$ for each node i in the network. The quality of fit of the role assignment to the image graph is then given by,

$$\mathcal{Q}^B(\{\sigma\}) = \frac{1}{M}\left(\sum_{i \neq j} a_{ij}A_{ij}B_{\sigma_i\sigma_j} + b_{ij}(1 - A_{ij})(1 - B_{\sigma_i\sigma_j})\right).$$

So the matching of an edge ij in the network ($A_{ij} = 1$) with an edge between the roles σ_i and σ_j in the image graph ($B_{\sigma_i\sigma_j} = 1$) is rewarded with some contribution a_{ij}. Similarly, matching non-edges are rewarded with a contribution b_{ij}.

By discarding terms not depending on σ and introducing the abbreviations

$$e_{rs} = \frac{1}{M}\sum_{i \neq j}(a_{ij} + b_{ij})A_{ij}\delta_{\sigma_i,r}\delta_{\sigma_j,s} \qquad \text{and}$$

$$[e_{rs}] = \frac{1}{M}\sum_{i \neq j} b_{ij}\delta_{\sigma_i,r}\delta_{\sigma_j,s}$$

we have,

$$\mathcal{Q}^B(\{\sigma\}) = \sum_{r,s}^{q}(e_{rs} - [e_{rs}])B_{rs} \qquad [1]$$

$$= -\sum_{r,s}^{q}(e_{rs} - [e_{rs}])(1 - B_{rs}) + C \qquad [2]$$

where $C = \sum_{r,s}^{q}(e_{rs} - [e_{rs}])$ is constant and we note the change from summing over nodes i and j to roles r and s.

Thus far the measure has been dependent on a choice of image graph, which is undesirable, and a comparison across all possible image graphs is unfeasible. Note however that from equations [1] and [2], we see that the optimal fit of a network to image graph is characterised by $e_{rs} - [e_{rs}] \geqslant 0$ for $B_{rs} = 1$ and by $e_{rs} - [e_{rs}] \leqslant 0$ for $B_{rs} = 0$. We can therefore aim to maximise the quality function

$$\mathcal{Q}^*(\sigma) = \frac{1}{2}\sum_{r,s}^{q}|e_{rs} - [e_{rs}]|.$$

If required we can then read off the image graph by taking $B_{rs} = 1$ when $e_{rs} - [e_{rs}] \geqslant 0$ and $B_{rs} = 0$ when $e_{rs} - [e_{rs}] \leqslant 0$.

Also considered in the paper is the fact that $\mathcal{Q}^*$ is inherently increasing in the number of roles q. So a comparison of $\mathcal{Q}^*$ with a maximum value $\mathcal{Q}^{max}$ is necessary if Q^* is to be used as an indicator for the optimal number of roles.

The authors used this method to analyse the roles of countries within the context of world trade.

We will not implement this method as an algorithm for role detection, but we will later use Q^* to compare our two principal algorithms. In doing so we will take $b_{ij} = (k_i^{out}k_j^{in})/M$ and $a_{ij} = 1 - b_{ij}$ as discussed in the paper, where k_i^{in} and k_i^{out} denote the in/out degrees of the node i.

Fitting to a Stochastic Block Model. Another possible approach is fitting a stochastic block model. That is, assuming that the underlying network follows a stochastic block model, and then using maximum likelihood estimation to try and find the parameters which best fit the data. See for example (7) or (8). In the context of community detection, this approach is often simplified by using a *Planted Partition Model* (8), a simplification of the Stochastic Block Model, where the entries of the probability matrix take one of two values, p_{in} for an intra-community edge or p_{out} for an edge between different communities. This simplification is not appropriate if the desire is to detect more general block structures.

While the methods we investigate in the next section have drawbacks of their own, they are more easily implemented numerically than those we have just discussed.

Pairwise Node Similarity Measures

We now turn to the main focus of this report. We describe and analyse two role detection based on pairwise node similarity measures. We could start as follows (9). Given a network and a representation of some role partition as an image graph, we could measure the *similarity* between each node in the network and its corresponding node in the image graph, according to some criteria (for example their connectivity patterns). We could then add the node in the network to the role class corresponding to the node to which it is most similar in the image graph. However, we do not want to have to predetermine the image graph ourselves. Instead, we consider the pairwise similarity between nodes in the original network, and then cluster nodes which are similar to each other. Our two methods take the following form:

- Decide what it means for two nodes to be "similar".
- Define a metric to quantify the similarity between any two nodes.
- Perform cluster analysis on the matrix of pairwise similarities.
- These clusters form the role classes.

Leicht, Holme and Newman

The first method is that of Leicht, Holme and Newman (4), which applies to undirected graphs. For convenience we may refer to this method as "LHN".

This method is based on the premise that a node i is similar to a node j if i has a neighbour v which is similar to j. Likewise j is similar to i if j has a neighbour which is similar to i. Despite the apparent asymmetry, this will lead to a measure which is perfectly symmetric: the similarity between i and j will be the same as that between j and i. This self-referential concept is reminiscent to that of regular equivalence. The

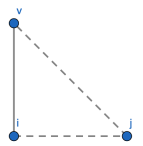

Fig. 4. Node i is similar to j if i has a neighbour v which is similar to j.

recursive description will require some starting point and so for that we make the natural claim that nodes are similar to themselves. Denoting the similarity between nodes i and j by S_{ij}, our measure of similarity will thus have contributions from the "neighbour term" and the self-similarity:

$$S_{ij} = \phi \sum_{v} A_{iv} S_{vj} + \psi \delta_{ij}$$

where $\mathbf{A} = (A_{ij})$ is the adjacency matrix of the network, and ϕ and ψ are parameters which govern the balance between the two contributions.

Letting $\mathbf{S} = (S_{ij})$, we have in matrix form,

$$\mathbf{S} = \phi \mathbf{A} \mathbf{S} + \psi \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix, and so,

$$\mathbf{S} = \psi [\mathbf{I} - \phi \mathbf{A}]^{-1}.$$

We see that ψ contributes just an overall multiplicative factor, and so since we are only interested in relative similarities between the node pairs, we set $\psi = 1$,

$$\mathbf{S} = [\mathbf{I} - \phi \mathbf{A}]^{-1}. \qquad [3]$$

Considering the similarity from the perspective of j being similar to i if j has a neighbour which is similar to i, we start with $S_{ij} = \phi \sum_v S_{iv} A_{vj} + \psi \delta_{ij}$ which leads to the same expression $\mathbf{S} = [\mathbf{I} - \phi \mathbf{A}]^{-1}$. Thus the definition provides a unique value for the similarity between i and j and is given by the element S_{ij} of the symmetric matrix $\mathbf{S}$. What's left is to choose ϕ.

By expanding [3] as a power series,

$$\mathbf{S} = \mathbf{I} + \phi \mathbf{A} + \phi^2 \mathbf{A}^2 + \ldots \qquad [4]$$

and since the element $[\mathbf{A}^l]_{ij}$ gives the number of walks of length l between i and j, we see that we get a contribution of ϕ^l to the similarity S_{ij} for each length l walk between i and j. However, some pairs of nodes, for example those with high degree, are *expected* to have more walks between them. We want to make some sort of correction/normalisation so that we reward the presence of a high number of walks between two nodes *relative* to how many we would expect by random chance. (Note that in a similar setting, our next algorithm does not make this consideration.) We choose the notion of "how many we would expect" to be the number of such walks in the configuration model with the same degree sequence.

We generalise [4] by allowing an independent coefficient C_l^{ij} for each term and node pair:

$$S_{ij} = \sum_{l=0}^{\infty} C_l^{ij} [\mathbf{A}^l]_{ij}. \qquad [5]$$

Letting N_l^{ij} be the expected number of walks of length l between i and j in the configuration model, an appropriate normalisation would be setting $C_l^{ij} = 1/N_l^{ij}$. However, calculations of N_l^{ij} become complicated, and so the authors settle for an approximation. This leads to values of,

$$C_0^{ij} = \delta_{ij} \quad \text{and} \quad C_l^{ij} = \frac{2m}{k_i k_j} \lambda_1^{1-l} \quad \text{for} \quad l \geqslant 1,$$

where m is the the number of edges in the network, k_i denotes the degree of node i and λ_1 is the largest eigenvalue of the adjacency matrix $\mathbf{A}$.

Unfortunately, substitution of these C_l^{ij} into [5] does not lead to a convergent sequence. To deal with this, the parameter $\alpha \in (0,1)$ is introduced. Varying α weights the significance of having more long walks than expected or having more short walks than expected. The authors claim that in general an appropriate value is $\alpha = 0.97$. Disregarding a term which affects only the diagonal elements of $\mathbf{S}$ (the node self-similarities which we do not tend to be interested in), we arrive at our final expression for the similarity,

$$S_{ij} = \frac{2m}{k_i k_j} \left[\left(\mathbf{I} - \frac{\alpha}{\lambda_1} \mathbf{A} \right)^{-1} \right]_{ij}.$$

Or in matrix form,

$$\mathbf{S} = 2m\lambda_1 \mathbf{D}^{-1} \left(\mathbf{I} - \frac{\alpha}{\lambda_1} \mathbf{A} \right)^{-1} \mathbf{D}^{-1}$$

where $\mathbf{D} = (D_{ij})$ is the diagonal matrix of degrees, $D_{ij} = k_i \delta_{ij}$.

We now use the similarity matrix to determine the role classes by performing spectral cluster analysis.

Testing the method. The authors first test their method on a computer generated network. The network is meant to represent relations between people of various ages, with relations more likely between those who are closer in age. The hope is that the algorithm finds the age groups.

We have produced a similar (but smaller) network using the stochastic block model with 5 blocks of 10 nodes each and the following matrix of probabilities:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{10} & \frac{1}{25} & \frac{1}{50} & \frac{1}{100} \\ \frac{1}{10} & \frac{1}{2} & \frac{1}{10} & \frac{1}{25} & \frac{1}{50} \\ \frac{1}{25} & \frac{1}{10} & \frac{1}{2} & \frac{1}{10} & \frac{1}{25} \\ \frac{1}{50} & \frac{1}{25} & \frac{1}{10} & \frac{1}{2} & \frac{1}{10} \\ \frac{1}{100} & \frac{1}{50} & \frac{1}{25} & \frac{1}{10} & \frac{1}{2} \end{pmatrix}$$

We can see from Fig. 5 that the algorithm does a good job at finding the age groups. The five "true" age groups are nodes 0 to 9, 10 to 19, 20 to 29, 30 to 39 and 40 to 49. So only the nodes 30 and 45 have been placed incorrectly.

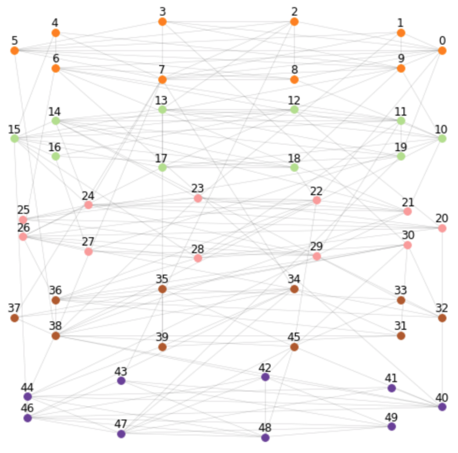

Fig. 5. The role classes found by the LHN algorithm on a contrived data set representing interactions between individuals of different ages. Each colour represents a role class.

Cooper and Barahona

Our second method is that of Cooper and Barahona (2), which we may refer to as "CB". It is designed for directed networks and is based on the concept of "flows". If we think of the directed edges of a network as describing the flow of some quantity through the nodes, we say two nodes are similar if they have similar flow environments; the pattern of incoming and outgoing flow is similar in the two nodes. This "pattern" is in fact entirely determined by the number of incoming and outgoing walks of each given length.

The similarity measure is defined as follows. Given a directed network, let N be the number of nodes and $\mathbf{A}$ its adjacency matrix, which for an directed network is in general asymmetric.

The number of outgoing walks of length k from a node i is given by the ith coordinate of the vector $\mathbf{A}^k\mathbf{1}$, where $\mathbf{1}$ is the $N \times 1$ vector of ones. Similarly, the number of incoming walks of length k into i is given by the ith coordinate of $(\mathbf{A}^\top)^k\mathbf{1}$.

We now construct a matrix $\mathbf{X}$ by appending the column vectors $\beta^k \mathbf{A}^k \mathbf{1}$ and $\beta^k (\mathbf{A}^\top)^k \mathbf{1}$ for lengths k up to some k_{max}, where $\beta = \frac{\alpha}{\lambda_1}$. The constant $\alpha \in [0,1]$ allows for tuning of the significance of long/short walks as in LHN.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} = \Big[\underbrace{\cdots (\beta \mathbf{A}^\top)^k \cdots}_{k_{max}} \Big| \underbrace{\cdots (\beta \mathbf{A})^k \cdots}_{k_{max}} \Big]$$

The row vector $\mathbf{x}_i$ gives the *flow profile* of the node i. It contains the (scaled) number of outgoing and incoming walks of each length $k \leqslant k_{max}$ to and from node i. Our measure is based on the fact that we consider i similar to j if they have similar flow profiles. To quantify this, we use the cosine distance, which gives a value close to one for similar vectors, and close to zero for dissimilar vectors.. We therefore obtain a similarity matrix $\mathbf{Y} = (Y_{ij})$, where the similarity between i and j is given by the cosine difference of the flow profiles of the two nodes.

$$Y_{ij} = \frac{\mathbf{x}_i \mathbf{x}_j^\top}{\|\mathbf{x_i}\| \|\mathbf{x_j}\|}$$

As before, given the similarity matrix, we perform cluster analysis to determine the role classes.

Testing the method. As previously, we use a stochastic block model to generate a first example on which to test the algorithm. We have 3 blocks and the following matrix of probabilities:

$$\begin{pmatrix} 0 & 0.9 & 0 \\ 0 & 0.4 & 0.9 \\ 0 & 0 & 0 \end{pmatrix}.$$

Each block contains 6 nodes. Note the obvious "flow" from the first to the second to the third block. We see that our algorithm indeed uncovers these blocks (Fig. 6).

As a second example, we carry out the algorithm on the "St Marks River food web" (10), where directed edges represent the flow of carbon. (This is the same data set as used in (2), and is exactly the sort of network the algorithm was designed to treat.) We see that the role classes found by the algorithm line up fairly well with the trophic levels of the system (Fig. 12 in the SI). At the top we see birds and predatory fish, then smaller fish and predatory invertebrates such as crab and shrimp. At the bottom we have carbon producers such as algae and bacteria. Some organisms such as the benthic algae and omnivorous crabs have been placed incorrectly by the algorithm.

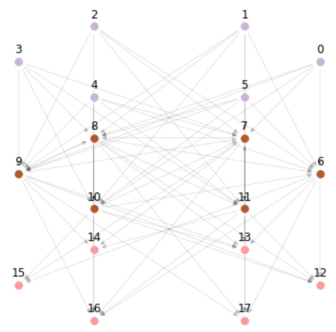

Fig. 6. The CB algorithm successfully distinguishes the role classes in a contrived flow network.

Comparison

In this section we try to compare the performance of the two algorithms. Despite being designed for directed networks, the CB algorithm is still valid and useful on undirected networks, and so we use undirected networks for the comparison. Using stochastic block models, we have generated sequences of networks with different role structures (Fig. 7).

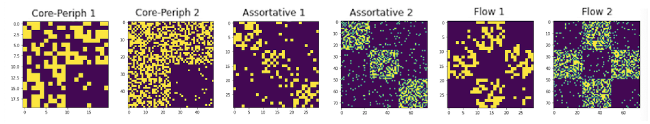

Fig. 7. Adjacency matrices showing different block structure networks with blocks of size 10 or 25.

For each structure, we have run the algorithms on 50 random stochastic block model networks and measured the quality of the found role classes using $\mathcal{Q}^*$. For each method and for each role structure, we have plotted the mean value of $\mathcal{Q}^*$ over the 50 random networks. For each structure, the first network has blocks of size 10 nodes and the second has block sizes of 25 nodes. Note that comparisons of $\mathcal{Q}^*$ are only valid between the partitions found by the two methods on the same network, not on the different networks.

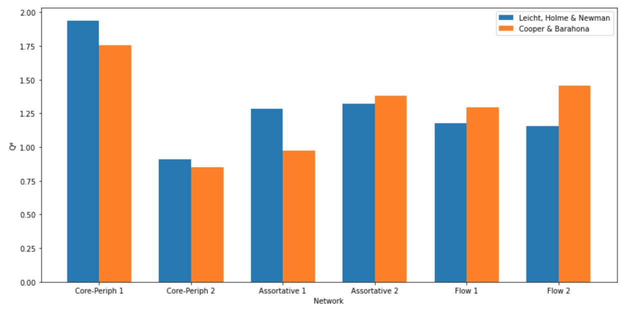

Fig. 8. Using $\mathcal{Q}^*$ to compare performance of LHN and CB algorithms on random networks with block structure.

Unsurprisingly, in the “flow” type of network, where there are connections between the first and the second, and between the second and the third blocks, the CB algorithm performed better according to the measure $\mathcal{Q}^*$. This is (an undirected version of) the type of network the CB algorithm was designed for.

It is perhaps surprising that the LHN algorithm didn't outperform the CB algorithm on both sizes of the assortative network, given its tendency to find community structure (see 'Critical Analysis' section).

Despite LHN apparently outperforming CB on the detection of a core-periphery structure, a closer look at the partitions obtained shows both methods to fare poorly in this situation (Fig. 10 in SI).

Critical Analysis

The *Operation Juanes* data set (11) is a network showing the interactions of individuals involved in cocaine trafficking in Spain. An application of the Louvain algorithm emphasises the community structure of the network (Fig. 9).

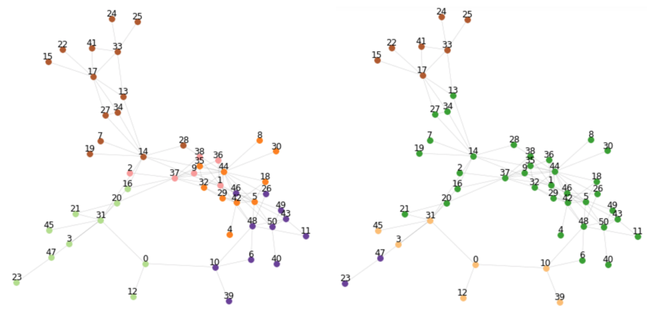

Fig. 9. Community structure in the Operation Juanes network (left) and the role classes found by the LHN algorithm (right).

When investigating the effectiveness of the LHN algorithm, the hope was that it might uncover some hierarchical structure which was shared by each community, perhaps highlighting leading figures in each one. However, even for different numbers of role classes, while it didn't produce the same partition, the LHN algorithm had a strong tendency to group together nodes from the same community.

When testing the LHN algorithm, we showed its relative success at differentiating the various *age groups*. However, in this example the age groups are more densely connected within themselves than with each other. What we have really done is just find the communities.

We also see (for example Fig. 13 in the SI) that in many examples the classes found by the method are very similar to the communities found by the Louvain algorithm.

The notion of similarity in this method (i being similar to j if i has a neighbour who is similar to j) means that two nodes can be similar even if they are not adjacent. However, the condition is still fairly “local”, and when we incorporate it into an algorithm, the effect is often to uncover something close to typical community structure.

One of the biggest shortcomings of both methods is their dependency on the number of role classes being specified in advance. This was fine when the network under consideration was one we created ourselves using the stochastic block model, and there may be situations when we might look for a specific number of roles. However, this would not be the case in general for analysis of real-world networks, and finding the number of roles makes up a significant part of uncovering the role structure of the network.

The paper (5) builds upon the method of Cooper and Barahona. They use a *Relaxed Minimum Spanning Tree* (RMST) algorithm to produce a network from the similarity matrix $\mathbf{Y}$, on which they consider a continuous-time Markov process. They uncover the role classes of the original network by using Markov stability to detect the communities of the RMST similarity network. A consequence of the time evolution is the *dynamic zooming* effect which allows for community detection at all levels of resolution, and hence does not impose a predetermined number of roles. Fig. 14 in the SI aptly demonstrates the application of this method. In this figure, we see how the dynamic zooming has allowed for detection of robust partitions of sizes 4, 3 and 2.

Another shortcoming of both methods is their failure to detect the evident role structure in a bipartite graph, see Fig. 11 in the SI.

Our methods group nodes into disjoint role classes. It may be the case that allowing nodes to assume multiple roles would be more appropriate. See for example (12) or in the context of community detection, (13).

Modelling issues and Empirical Data

We have tested and compared the performance of our algorithms on artificial networks, generated by the stochastic block model. There is no issue if the end goal is simply to see how well the methods reproduce the block structure which we fed into the SBM. However, we would like to infer from these results the performance of the algorithms on real-world networks. In doing so, we would be making the implicit modelling assumption that the underlying role structure of the real network was governed by some SBM. It is perceivable that a network could have some role structure which cannot be described by a SBM, whose capacity to describe such structure is limited by its parameters. To assume it governed by a SBM would be to neglect the intricacies of the underlying relationships governing the structure of the network. For example, a SBM network has one common value for the edge probability between any two nodes in the same class. This may be an inappropriate modelling assumption if the structure within a role class is significant.

The two empirical data sets we have used are those of the *St Marks River food web* and *Operation Juanes*. The edges in the Operation Juanes network represent interactions between individuals, where the information comes from wire tappings and meetings recorded in police investigations. The nature of this data means it may well be incomplete, it is unlikely that the police investigations make up the complete picture of interactions. This means that the network might not truly reflect the real-life situation. Although it wasn't the initial intention, our use of this network amounted to just providing an example of the LHN algorithm finding something close to communities, and so whether or not the network accurately reflects the real-life interactions is not important.

The data collection for the movement of carbon in the St Marks River food web is complex (10). Small uncertainties in the data are however unlikely to cause a significant discrepancy between the produced network and reality. Our analysis of the trophic levels to test the CB algorithm is reasonably credible.

Conclusion

In this report we have analysed two algorithms for role detection and shown their success and failures in various situations. In each case, the algorithms performed fairly well at detecting role classes which lined up with their respective definitions of similarity. It was perhaps unreasonable to hope that either algorithm would perform well in the setting of general role detection. For example, the CB algorithm is designed to detect classes of nodes with similar flow characteristics in networks where directed edges show the flow of some quantity. It is unfair to critique it on the fact that it failed to detect say, a core-periphery structure, when using a different algorithm designed for that specific purpose would have been more appropriate. In both cases, central to the algorithms were their definition of similarity, so the type of partition they would find was built in to their definition from the start.

Further investigation would start with a deeper examination of the extension to the CB algorithm (5), restricted to the case of flow networks.

Materials and Methods

The St Marks food web data set can be found here: http://vlado.fmf.uni-lj.si/pub/networks/data/bio/foodweb/foodweb.htm, and the Operation Juanes data set can be found here: https://sites.google.com/site/ucinetsoftware/datasets/covert-networks/cocainesmuggling.

ACKNOWLEDGMENTS.

1. Girvan M, Newman MEJ (2002) Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99(12):7821–7826.
2. Cooper K, Barahona M (2010) Role-based similarity in directed networks. *Computing Research Repository - CORR.*
3. Reichardt J, White DR (2007) Role models for complex networks. *The European Physical Journal B* 60(2):217–224.
4. Leicht EA, Holme P, Newman MEJ (2006) Vertex similarity in networks. *Physical Review E* 73(2).
5. Beguerisse-Diaz M, Vangelov B, Barahona M (2013) Finding role communities in directed networks using role-based similarity, markov stability and the relaxed minimum spanning tree. *2013 IEEE Global Conference on Signal and Information Processing.*
6. Rossi RA, Ahmed NK (2015) Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering* 27(4):1112–1131.
7. Peixoto TP (2014) Efficient monte carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E* 89(1).
8. Prokhorenkova L, Tikhonov A (2019) Community detection through likelihood optimization: In search of a sound model in *The World Wide Web Conference*, WWW '19. (Association for Computing Machinery, New York, NY, USA), p. 1498–1508.
9. Blondel V, Gajardo A, Heymans M, Senellart P, Van Dooren P (2004) A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Review* 46:647–666.
10. Baird D, Luczkovich J, Christian R (1998) Assessment of spatial and temporal variability in ecosystem attributes of the St Marks national wildlife refuge, Apalachee Bay, Florida. *Estuarine, Coastal and Shelf Science* 47(3):329–349.
11. Framis AGS (2011) *Illegal networks or criminal organizations: Power, roles and facilitators in four cocaine trafficking structures*, Third Annual Illicit Networks Workshop. (Montreal).
12. White A, Chan J, Hayes C, Murphy T (2012) *Mixed Membership Models for Exploring User Roles in Online Fora*, Sixth International AAAI Conference on Weblogs and Social Media. (Dublin).
13. Huang W, Liu Y, Chen Y (2020) Mixed Membership Stochastic Blockmodels for Heterogeneous Networks. *Bayesian Analysis* 15(3):711 – 736.

Supporting Information (SI)

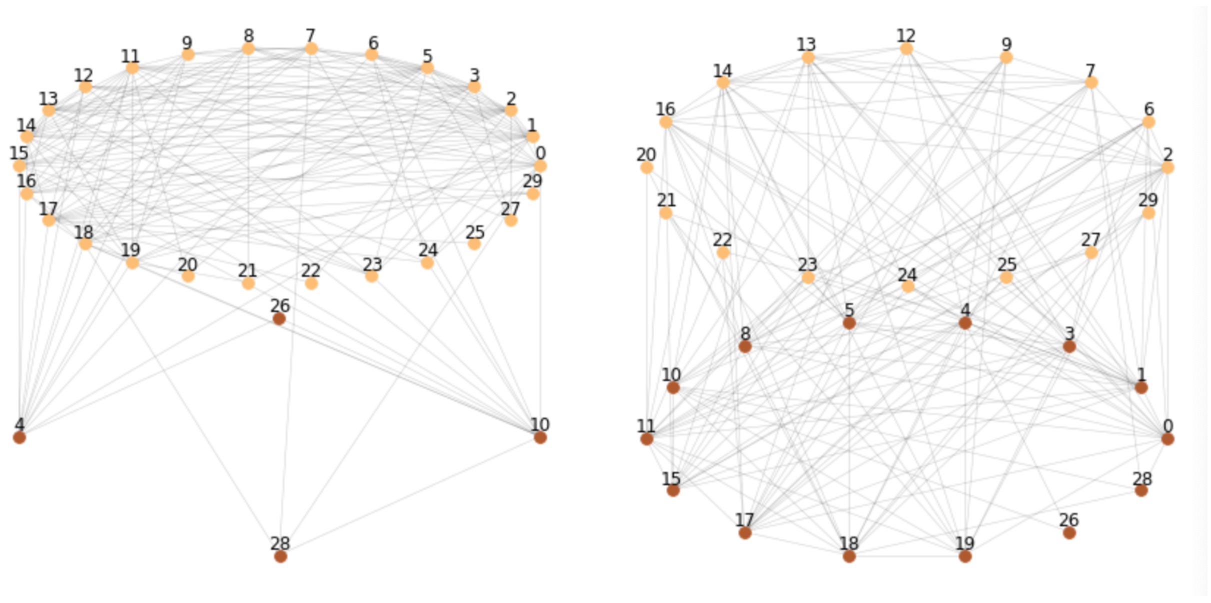

Fig. 10. Both algorithms perform poorly at detecting core-periphery structure. The correct partition would have one class containing nodes 0 to 19 and the other containing 20 to 29. The role classes produced by the LHN algorithm is shown on the left and by CB on the right.

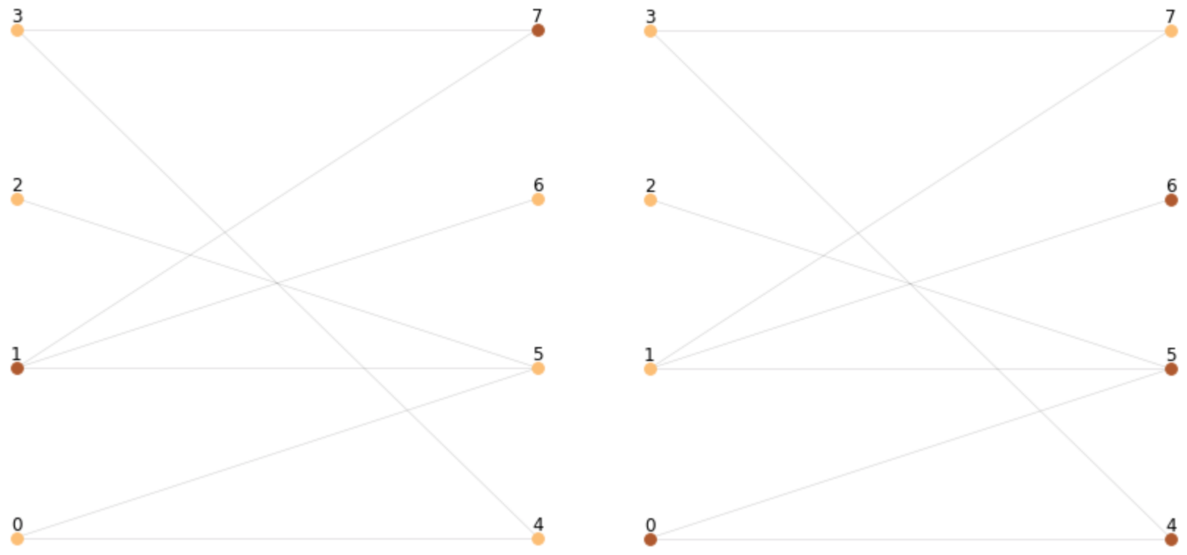

Fig. 11. Neither the LHN (left) nor the CB (right) algorithm is able to detect bipartite structure. In each network, the colours represent the role classes found by the respective algorithms.

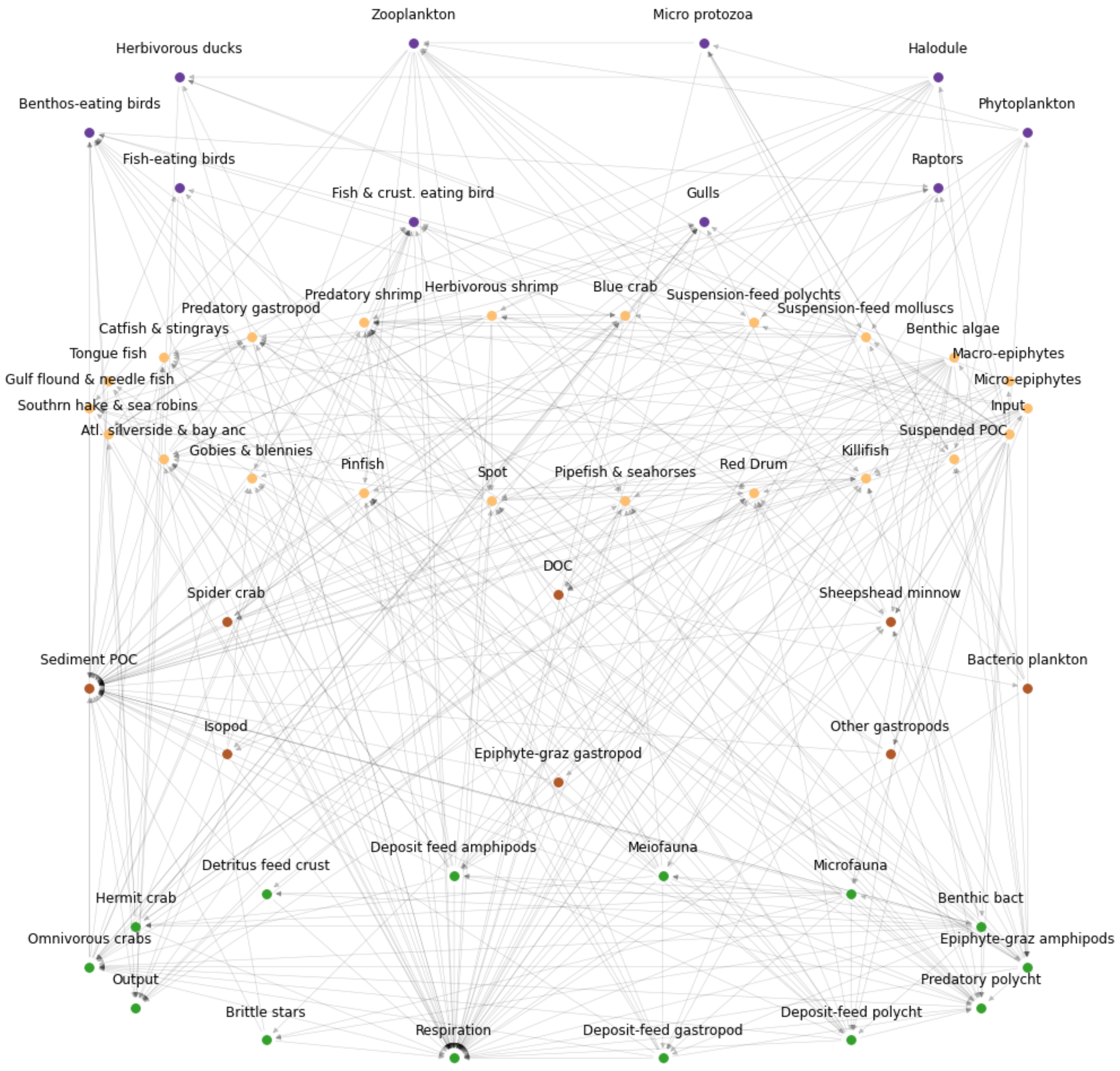

Fig. 12. The role classes found by the CB algorithm roughly coincide with trophic levels of the St Marks foodweb.

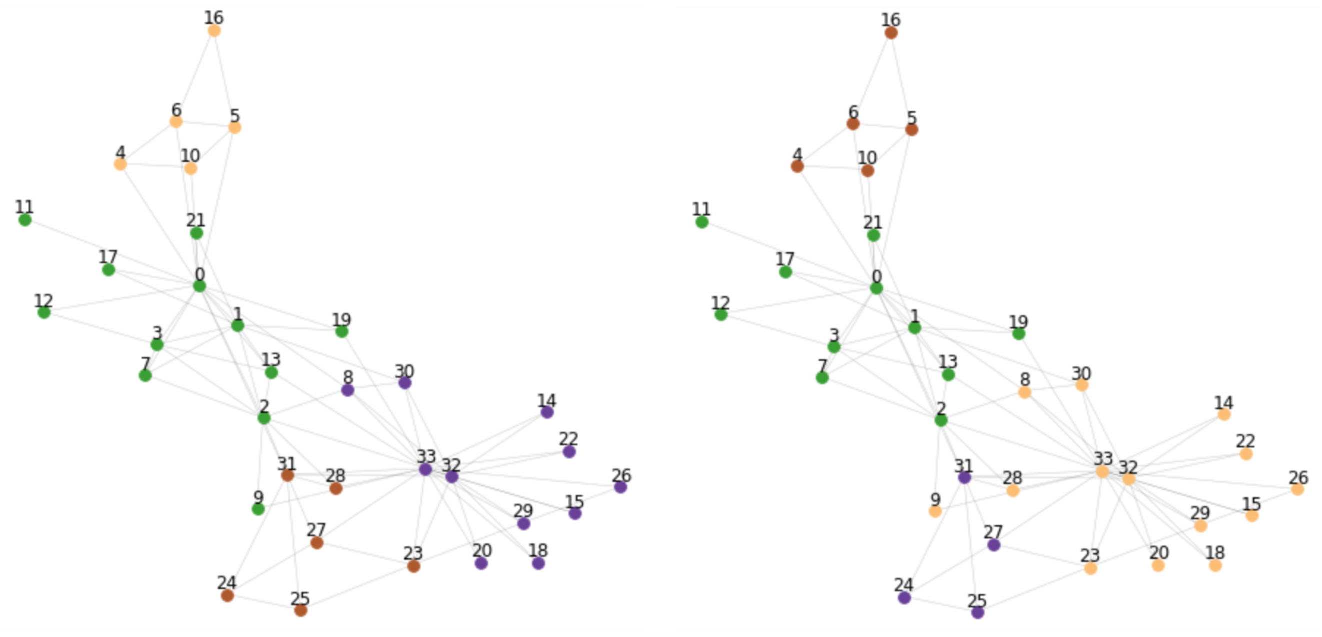

Fig. 13. The Karate Club network. An example of the LHN algorithm (right) producing similar results to the Louvain community detection algorithm (left).

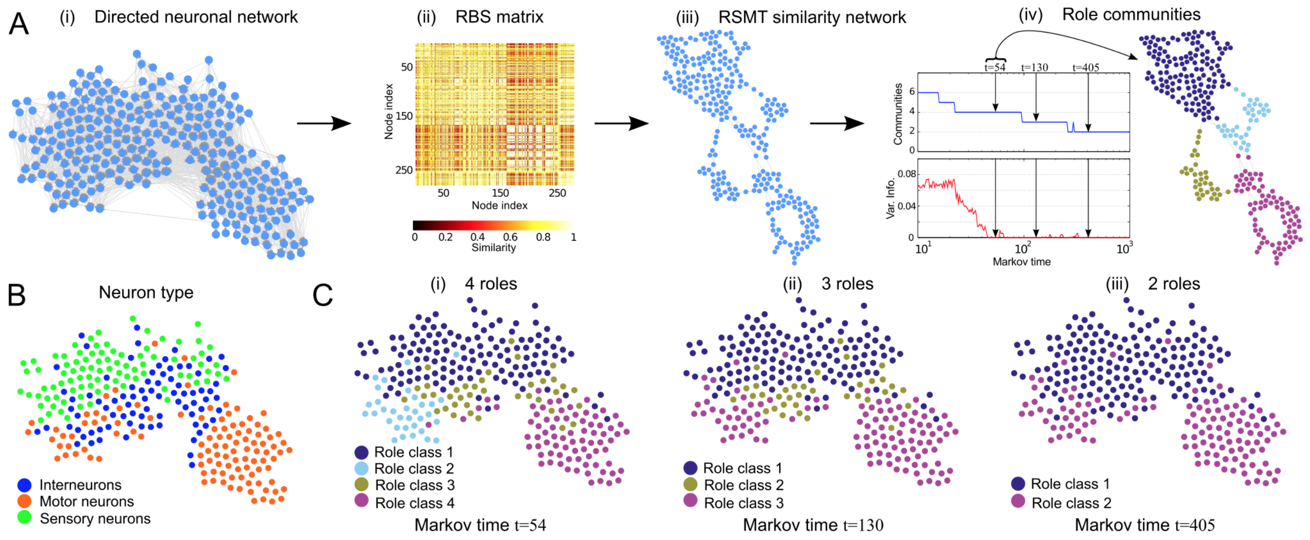

Fig. 14. Demonstration of the extension of CB using Markov stability. This figure is taken directly from (5).

Code. The following is Python code for the main functions used in the simulations. The rest can be found in the Jupyter notebook.

```

def Cooper_Barahona(A,kmax,alpha=0.9):

    '''Takes adjacency matrix A and returns the similarity matrix Y
    of the flow profiles of each node according to the method of
    Cooper and Barahona'''

    N = len(A)                                  # number of nodes
    lambda1 = max(np.linalg.eigvals(A))         # largest e-value
    beta = alpha/lambda1

    At = np.transpose(A)
    X = At.copy()
    I = np.ones((N,1))
    X = beta*np.matmul(X,I)


    for k in range(2,kmax+1):
        X = np.concatenate((X,(beta**k)*np.matmul(np.linalg.matrix_power(At,k),I)),axis=1)

    for k in range(1,kmax+1):
        X = np.concatenate((X,(beta**k)*np.matmul(np.linalg.matrix_power(A,k),I)),axis=1)

    Y = np.zeros((N,N))

    for i in range(N):
        Y[i] = [(1 - scipy.spatial.distance.cosine(X[i],X[j])) for j in range(N)]

    return Y

def L_H_N(G,alpha=0.97):

    '''Takes graph G and returns similarity matrix S
    of the nodes according to Leicht, Holme and Newman's
    definition of similarity'''

    n = G.number_of_nodes()
    m = G.number_of_edges()
    A = nx.convert_matrix.to_numpy_matrix(G)          #adjacency matrix
    lambda1 = np.real(max(np.linalg.eigvals(A)))      #largest e-value
    degrees = list(G.degree)
    degrees = [degrees[i][1] for i in range(len(degrees))]
    D = np.diag(degrees)
    Dinv = np.linalg.inv(D)
    M = np.identity(n) - (alpha/lambda1)*A
    Minv = np.linalg.inv(M)

    S = 2*m*lambda1*np.matmul(np.matmul(Dinv,Minv),Dinv)

    return S

def spectral_clustering(S,n_clusts):

    clustering = SpectralClustering(n_clusters=n_clusts,assign_labels="discretize",
                                    affinity='precomputed').fit(S)

    partition = [[] for i in range(n_clusts)]

    for i in range(len(clustering.labels_)):
        partition[clustering.labels_[i]].append(i)

    return partition

def Visualise_roles(G,partition,order=None,spacing=0.3,labelheight=0.3,labels=None): #reference PS3

    '''Takes a graph and a partition of its nodes into classes
    (as a list of lists of nodes in each class) and displays the
    classes separately and in different colours'''

    if order is None:
        order = list(range(len(partition)))

    plt.figure (figsize=(7,7))

    size = len(partition)

    #This line of code is from the solutions to PS3
    cm = plt.cm.ScalarMappable(cmap=plt.get_cmap('Paired'),
                               norm=mpl.colors.Normalize(vmin=0, vmax=float(size),
                                                         clip=False))

    C = [nx.circular_layout(G.subgraph(partition[i])) for i in order]

    #separating the classes in the partition

    for i in range(1,size):
        d = spacing + max([y for x,y in C[i].values()]) - min([y for x,y in C[i-1].values()])

        for j in C[i].keys():
            C[i][j][1] = C[i][j][1] - d

    pos = {}

    for i in range(size):
        pos.update(C[i])

    #This loop is from the solutions to PS3
    count = 0
    for i in range(size):
        count = count + 1.
        nx.draw_networkx_nodes(G, pos, partition[i], node_size=60, node_color=[cm.to_rgba(count)])

    nx.draw_networkx_edges(G, pos, alpha=0.2, width = 0.5)

    for i in pos.keys(): #making labels above nodes
        pos[i][1] += labelheight

    if labels is None:
        pos2 = pos
    else:
        pos2 = copy.copy(labels)
        pos2 = {value:key for key, value in pos2.items()}

        j = 0
        for i in pos2.keys():
            pos2[i] = pos[j]
            j += 1

        nx.relabel_nodes(G, labels, copy=False)

    nx.draw_networkx_labels(G, pos2)

    plt.show()
    return partition

def Visualise(G,partition,labelheight=0.3):

    '''Takes a graph and a partition of its nodes into classes and displays the
    classes in different colours'''

    plt.figure (figsize=(6,6))

    size = len(partition)

    #This line of code is from the solutions to PS3
    cm = plt.cm.ScalarMappable(cmap=plt.get_cmap('Paired'),
                               norm=mpl.colors.Normalize(vmin=0, vmax=float(size),
                                                         clip=False))

    pos = nx.spring_layout(G)

    #This loop is from the solutions to PS3
    count = 0
    for i in range(size):
        count = count + 1.
        nx.draw_networkx_nodes(G, pos, partition[i], node_size=60, node_color=[cm.to_rgba(count)])

    nx.draw_networkx_edges(G, pos, alpha=0.2, width = 0.5)

    for i in pos.keys(): #making labels above nodes
        pos[i][1] += labelheight

    nx.draw_networkx_labels(G, pos)
    plt.show()
    return partition

def Qstar(G,partition):

    '''Takes a graph and a partition of its nodes and calculates Q*
    as defined by Reichardt and White'''

    M = G.number_of_edges()
    A = nx.convert_matrix.to_numpy_matrix(G)     #adjacency matrix
    A = np.asarray(A)
    q = len(partition)       #number of classes in partition

    if nx.classes.function.is_directed(G) is True:
        in_degs = list(dict(G.in_degree()).values())
        out_degs = list(dict(G.out_degree()).values())
    else:
        in_degs = list(dict(G.degree()).values())
        out_degs = list(dict(G.degree()).values())

    K_ins = [0]*q
    for r in range(q):
        K_ins[r] = sum([in_degs[j] for j in partition[r]])

    K_outs = [0]*q
    for r in range(q):
        K_outs[r] = sum([out_degs[j] for j in partition[r]])

    S = [[0]*q]*q

    for r in range(q):
        for s in range(q):
            S[r][s] = sum([sum([A[i][j] for j in partition[s]]) for i in partition[r]])

    for r in range(q):
        S[r][r] = S[r][r] - sum(A[i][i] for i in partition[r])

    Q = 0.5*sum([sum([(1/M**2)*abs(M*S[r][s] - (K_outs[r]*K_ins[s])) for s in range(q)]) for r in range(q)])

    return Q
```