MASTER'S THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: On methods to assess the significance of community structure in networks of financial time series

Author: Martí Renedo Mirambell

Advisor: Argimiro Arratia Quesada

Department: Department of Computer Science & BGSMath

Academic year: 2017

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering

Master's thesis

On methods to assess the significance of community structure in networks of financial time series.

Martí Renedo Mirambell

Supervised by Argimiro Arratia Quesada
Department of Computer Science & BGSMath

June, 2017

Thanks to Argimiro for helping, giving new ideas and supervising this project.

Abstract

We consider the problem of determining whether the community structure found by a clustering algorithm applied to financial time series is statistically significant, when no other information than the observed values and a similarity measure among time series is available. As a subsidiary problem we also analyze the influence of the choice of similarity measure in the accuracy of the clustering method.

We propose two raw-data based methods for assessing robustness of clustering algorithms on time-dependent data linked by a relation of similarity: One based on community scoring functions that quantify some topological property that characterizes ground-truth communities, the other based on random perturbations and quantification of the variation in the community structure. These methodologies are well-established in the realm of unweighted networks; our contribution are versions of these methodologies properly adapted to complete weighted networks.

We reinforce our assessment of the accuracy of the clustering algorithm by testing its performance on synthetic ground-truth communities of time series built through Monte Carlo simulations of VARMA processes.

Keywords

clustering, financial time series, ground-truth communities, similarity measures, Forex network

1. Introduction

We treat in this work the problem of determining the intrinsic structure of clustered data, where the clusters are based on some measure of similarity affecting all pairs of data points. From a network analysis perspective we are concerned with assessing the significance of communities formed by some unsupervised classification algorithm (i.e. clustering procedure) applied to fully-connected weighted networks.

We are motivated by research in community structure and their dynamics in financial market networks, characterize by a fixed number of nodes, each representing a financial time series, and links among all pairs of nodes weighted by the values of a measure of similarity, commonly based on pairwise correlation, between pairs of time series (see, e.g., [1],[2],[3],[4],[5]). In our previous work [6] we presented empirical evidence of the impact of the chosen similarity measure on the clustering results: In a foreign exchange (Forex) network, and clustering based on the Girvan-Newman modularity maximization algorithm [7, 8], we analyzed the qualitative differences in the clusterings obtained under three different correlation measures: Pearson, Kendall and the most recent distance correlation [9]. As an application of the statistical and topological criteria that we developed and present here to assess robustness of clustering on weighted networks, we shall give quantitative measures of the nature of the clustering obtained by considering similarity either based on Pearson or on distance correlation.

To assess the significance of communities structure in complete weighted networks, we developed a collection of cluster scoring functions that measure some topological characteristic of the ground-truth communities as defined by Yang and Leskovec in [10] for unweighted networks. Our scoring functions are proper adaptation of theirs to weighted networks. We then combined these topological measures of robustness of clusters with an analysis of the variation of successive random perturbations of the original network. The perturbations consist on changing the weights distribution and in an statistical sense degenerate the original network, and variation is measure in terms of the change of information (in the sense of Shannon's Theory of Information [11]). The idea is that a robust community should differ in its structural properties from the random perturbations inasmuch as these affect greater proportions of the network.

As a final check of clustering performance we construct synthetic networks with known ground-truth communities with respect to correlation using a methodology from [12] based on Monte Carlo simulations of VARMA processes. A good clustering algorithm should consistently detect the ground-truth communities of time series following an ARMA model.

2. Basic definitions

2.1 The Forex Network

The networks of exchange rates studied in [6][1] are built by considering the exchange rates as vertices and drawing edges between these vertices, weighted by the similarity between the returns of the pair of chosen exchange rates. We will focus on two possible similarity measures: one based on the Pearson correlation and the other based on the distance correlation[9].

For the Pearson similarity network, the adjacency matrix is defined as

$$A^{\rho}_{ij} = \frac{1}{2}(\rho(r^i, r^j) + 1) - \delta_{ij}. \tag{1}$$

This scales the Pearson correlation from $[-1, 1]$ to $[0, 1]$, while the Kronecker delta δ_{ij} removes self-edges.

In the graph with adjacency matrix A^{ρ} exchange rates with positively linearly correlated returns will be connected by edges of weight close to 1, and weight near 0 if the correlation is negative. Edges connecting non correlated exchanges will have weights closer to the center of the interval $[0, 1]$.

In the case of the distance correlation, the network is simply built from the matrix of distance correlations, $A^{\mathcal{R}}$ by removing self edges. For each pair of exchange rate returns r^i, r^j,

$$A^{\mathcal{R}}_{ij} = \mathcal{R}(r^i, r^j) - \delta_{ij} \tag{2}$$

2.2 Community Detection

The partition of the networks into communities is done using the Potts method. It consists on minimizing an objective function, the Potts Hamiltonian, which evaluates the strength[1] of a partition of the graph. This can be seen as a generalization of the modularity function[7].

Definition 2.1. The modularity of the partition $\mathcal{P}$ of a weighted undirected graph with adjacency matrix A is given by

$$Q(\mathcal{P}) = \frac{1}{2m} \sum_{ij} [A_{ij} - P_{ij}] \delta(c_i, c_j) \tag{3}$$

where c_i is the community of the node i in the partition $\mathcal{P}$ (so $\delta(c_i, c_j)$ is 1 when i and j are in the same community and 0 otherwise), P_{ij} is the expected weight of the edge ij in a null model and m is the sum of the weights of all edges in the graph.

Definition 2.2. The Hamiltonian of the Potts system of the partition $\mathcal{P}$ of a weighted undirected graph with adjacency matrix A is given by

$$H(\mathcal{P}) = -\sum_{ij} [A_{ij} - \gamma P_{ij}] \delta(c_i, c_j)$$

where γ is a parameter which determines how likely vertices are to form communities.

The algorithm used to minimize the Potts Hamiltonian has been adapted from the modularity maximization algorithm in [8] to suit weighted networks and this objective function.

3. Cluster scoring functions

Here we will provide functions which will evaluate the division of networks into clusters, specifically when the edges have weights. Using the scoring functions for communities in unweighted networks given in [10] as a reference, we propose generalizations to extend them to the weighted case.

Basic definitions

Let $G(V, E)$ be an undirected graph of order $n = |V|$ and size $m = |E|$. In the case of a weighted graph $\tilde{G}(V, \tilde{E})$[2], we will denote $\tilde{m} = \sum_{e \in \tilde{E}} w(e)$ the sum of all edge weights. Given $S \subset G$ a subset of vertices of the graph, we have $n_S = |S|$, $m_S = |\{(u, v) \in E : u \in S, v \in S\}|$, and in the weighted

[1] Considering a strong partition one that has strong links inside the communities and weak links between them.

case $\tilde{m}_S = \sum_{(u,v)\in\tilde{E}:u,v\in S} w((u,v))$. Note that if we treat an unweighted graph as a weighted graph with weights 0 and 1 (1 if two vertices are connected by an edge, 0 otherwise), then $m = \tilde{m}$ and $m_S = \tilde{m}_S$ for all $S \subset V$.

The following definitions will also be needed later on:

- $c_S = |\{(u,v) \in E : u \in S, v \notin S\}|$ is the number of edges connecting S to the rest of the graph.
- $\tilde{c}_S = \sum_{(u,v)\in E:u\in S,v\notin S} w_{uv}$ is the natural extension of c_S to weighted graphs; the sum of weights of all edges connecting S to $G \setminus S$.
- $\tilde{d}(u) = \sum_{v\neq u} w_{uv}$ is the natural extension of the vertex degree $d(u)$ to weighted graphs; the sum of weights of edges incident to u.
- $d_S(u) = |\{v \in S : (u,v) \in E\}|$ and $\tilde{d}_S(u) = \sum_{v\in S} w_{uv}$ are the (unweighted and weighted, respectively) degrees[3]restricted to the subgraph S.
- d_m and $\tilde{d}_m$ are the median values of $d(u)$, $u \in V$.[4]

Scoring functions

The left column in table 1 shows the community scoring functions for unweighted networks defined in [10]. These functions characterize some of the properties that are expected in networks with a strong community structure, with more ties between nodes in the same community than connecting them to the exterior. There are scoring functions based on internal connectivity (internal density, edges inside, average degree, fraction over median degree, triangle participation ratio), external connectivity (expansion, cut ratio) or a combination of both (conductance, normalized cut, and maximum, average and minimum out degree fractions).

On the right column we propose generalizations to the scoring functions which are suitable for weighted graphs while most closely resembling their unweighted counterparts. Note that for graphs which only have weights 0 and 1 (1 indicates that an edge exists, 0 that it doesn't) each pair of functions is equivalent (any definition that didn't satisfy this wouldn't be a generalization at all).

- **Internal density, edges inside, average degree:** These definitions are easily and naturally extended by replacing the number of edges (either inside the given community or adjacent to a given vertex) by the sum of their weights.
- **Fraction Over Median Degree (FOMD):** Fraction of vertices which have internal degree d_S higher than d_m, the median degree. It can also be extended easily to the weighted case using the corresponding definitions of degree and internal degree in Section 3. However, the financial networks we want to study have very high degrees in all vertices (which results in high d_m), so we can expect very few vertices to have internal degree (which is bounded by the size of the communities, in most cases small relative to the size of the graph) higher than that.

[2]For every variable or function defined over the unweighted graph, will use a "$\sim$" to denote its weighted counterpart

[3]We assume the weight function w_{uv} is defined for every pair of vertices u,v of the weighted graph, with $w_{uv} = 0$ if there is no edge between them.

[4]To prevent confusion between the function $d_S(\cdot)$ and the median value (which only depends on G) d_m we will always refer to subgraphs of G with uppercase letters.

Table 1: Community scoring functions for weighted and unweighted networks.

	unweighted	weighted
Internal density	$f(S) = \frac{m_S}{n_S(n_S-1)/2}$	$f(S) = \frac{\tilde{m}_s}{n_S(n_S-1)/2}$
Edges Inside	$f(S) = m_S$	$f(S) = \tilde{m}_S$
Average Degree	$f(S) = \frac{2m_S}{n_S}$	$f(S) = \frac{2\tilde{m}_S}{n_S}$
Fraction Over Median Degree	$f(S) = \frac{\lvert\{u \in S : d_S(u) > d_m\}\rvert}{n_s}$	$f(S) = \frac{\lvert\{u \in S : \tilde{d}_S(u) > \tilde{d}_m\}\rvert}{n_s}$
Triangle Participation Ratio	$f(S) = \frac{\lvert\{u \in S : \exists v,w \in S,\ (u,v),(u,w),(v,w) \in E\}\rvert}{n_S}$	-
Expansion	$f(S) = \frac{c_s}{n_s}$	$f(S) = \frac{\tilde{c}_s}{n_s}$
Cut Ratio	$f(S) = \frac{c_s}{n_s(n-n_s)}$	$f(S) = \frac{\tilde{c}_s}{n_s(n-n_s)}$
Conductance	$f(S) = \frac{c_s}{2m_s+c_s}$	$f(S) = \frac{\tilde{c}_s}{2\tilde{m}_s+\tilde{c}_s}$
Normalized Cut	$f(S) = \frac{c_s}{2m_s+c_s}$	$f(S) = \frac{\tilde{c}_s}{2\tilde{m}_s+\tilde{c}_s}$
Maximum ODF	$f(S) = \max_{u \in S} \frac{\lvert\{(u,v) \in E : v \notin S\}\rvert}{d(u)}$	$f(S) = \max_{u \in S} \frac{\sum_{v \notin S} w_{uv}}{\tilde{d}(u)}$
Average ODF	$f(S) = \frac{1}{n_s} \sum_{u \in S} \frac{\lvert\{(u,v) \in E : v \notin S\}\rvert}{d(u)}$	$f(S) = \frac{1}{n_s} \sum_{u \in S} \frac{\sum_{v \notin S} w_{uv}}{\tilde{d}(u)}$
Flake ODF	$f(S) = \frac{\lvert\{u \in S : \lvert\{(u,v) \in E : v \in S\}\rvert < d(u)/2\}\rvert}{n_s}$	$f(S) = \frac{\lvert\{u \in S : \sum_{v \in S} w_{uv} < \sum_{v \notin S} w_{uv}\}\rvert}{n_s}$

- **Triangle Participation Ratio (TPR):** This scoring function has not been considered because, given the discrete nature of its definition, no satisfactory extension into the weighted case was found.
- **Expansion:** Average number of edges connected to the outside of the community, per node. For weighted graphs, average sum of edges connected to the outside, per node.
- **Cut Ratio:** Fraction of edges leaving the cluster, over all possible edges. The proposed generalization is reasonable because edge weights are upper bounded by 1 and therefore relate easily to the unweighted case. In more general weigthed networks, however, this could take values well over 1 while lacking many "potential" edges (as edges with higher weights would distort the measure). In general bounded networks (with bound other than 1) it would be reasonable to divide the result by the bound, which would result in the function taking values between 0 and 1 (0 with all possible edges being 0 and 1 when all possible edges reached the bound).
- **Conductance and normalized cut:**Again, these definitions are easily extended using the methods described above.
- **Maximum and average Out Degree Fraction:** Maximum and average fractions of edges leaving the cluster over the degree of the node. Again, in the weighted case the number of edges is replaced by the sum of edge weights.
- **Flake Out Degree Fraction:** Fraction of nodes that have fewer edges pointing inside of the cluster than outside. In the weighted case, fraction of nodes with less edge weight pointing inside than outside.

Here, the modularity function (which is studied on [10]) has not been considered because the our networks have been split into communities with modularity-based optimization techniques, and therefore the results would be redundant. Additionally, the fraction over median degree and flake out degree fraction

have trivial values (0 and 1 respectively) in all of the networks (see table 5). This happens because in complete networks the amount of inter-comunity edges is very high compared to the intra-comunity edges. While this renders these scoring functions useless for the analysis of complete networks, they could be meaningful when studying more sparse weighted networks.

Clustering coefficient

Another possible scoring function for communities is the clustering coefficient or transitivity: the fraction of closed triplets over the number of connected triplets of vertices. A high internal clustering coefficient (computed on the graph induced by the vertices of a community) matches the intuition of a well connected and cohesive community inside a network, but its generalization to weighted networks is not trivial.

There have been several attempts to come up with a definition of the clustering coefficient for weighted networks. One of them (which, for instance, is the one implemented in the `igraph` function `transitivity`) was proposed in [13] and is given by $c_i = \frac{1}{\tilde{d}(i)(d(i)-1)} \sum_{j,h} \frac{w_{ij}+w_{ih}}{2} a_{ij} a_{jh} a_{ih}$. Note that this gives a local (*i.e.* defined for each vertex) clustering coefficient.

While this may work well on some weighted networks, in the case of complete networks, such as those built from correlation of time series,

$$c_i = \frac{1}{\tilde{d}(i)(d(i)-1)} \sum_{i,j} \frac{w_{ij}+w_{ih}}{2} = \frac{\sum_j w_{ij} + \sum_h w_{ih}}{\tilde{d}(i)(n-2)\cdot 2} = \frac{2\tilde{d}(i)}{\tilde{d}(i)(n-2)\cdot 2} = \frac{1}{n-2} \tag{4}$$

is constant on all edges and doesn't give any information about the network.

An alternative was proposed in [14] with complete weighted networks (with weights in the interval $[0,1]$) in mind, which makes it more adequate for our case:

- For $t \in [0,1]$ let A_t be the adjacency matrix with elements $a^t_{ij} = 1$ if $w_{ij} \geq t$ and 0 otherwise.
- Let C_t the clustering coefficient of the graph defined by A_t.
- The resulting weighted clustering coefficient is defined as

$$\tilde{C} = \int_0^1 C_t \, dt \tag{5}$$

Since C_t can only take as many different values as the number of different edge weights in the network, the integral is actually a finite sum. However, computing C_t (which is not computationally trivial) potentially as many as $n(n-1)$ times would be very costly for large values of n , so this function has been implemented by approximating the integral dividing the interval [0,1] into `n_step` parts (where `n_step`[5]is much smaller than $n(n-1)$).

It is worth noting that some of the introduced functions (internal density, edges inside, average degree, FOMD, clustering coefficient) take higher values the stronger the clusterings are, while the others (expansion, cut ratio, conductance, normalized cut and out degree fractions) do the opposite.

[5]In thist case we set `n_step`=100. This gives a reasonable resolution while keeping the computations fast.

3.1 Variation of information

To compare and measure how similar two clusterings of the same network are, we will use the variation of information; a criterion introduced in [11] and which is based on information theory.

Definition 3.1. The entropy of a partition $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_K\}$ of a set is given by:

$$\mathcal{H}(\mathcal{P}) = -\sum_{k=1}^{K} \frac{|\mathcal{P}_k|}{n} \log(\frac{|\mathcal{P}_k|}{n}), \tag{6}$$

where n is the size of the set and $\mathcal{P}_k$ is the k-th cluster of the partition.

Definition 3.2. Given $P(k, k') = \frac{|\mathcal{P}_k \cap \mathcal{P}'_{k'}|}{n}$ the joint probability distribution of elements belonging to clusters $\mathcal{P}_k$ and $\mathcal{P}_{k'}$, the mutual information is defined as:

$$I(\mathcal{P}, \mathcal{P}') = \sum_{k=1}^{K} \sum_{k'=1}^{K'} P(k, k') \log \frac{P(k, k')}{P(k)P'(k')} \tag{7}$$

Definition 3.3. The variation of information of partitions $\mathcal{P}$ and $\mathcal{P}'$ information is given by:

$$VI(\mathcal{P}, \mathcal{P}') = \mathcal{H}(\mathcal{P}) + \mathcal{H}(\mathcal{P}') - 2I(\mathcal{P}, \mathcal{P}') \tag{8}$$

Intuitively, the mutual information measures how much knowing the membership of an element of the set in partition $\mathcal{P}$ reduces the uncertainty of its membership in $\mathcal{P}'$. This is consistent with the fact that the mutual information is bounded between zero and the individual partition entropies

$$0 \leq I(\mathcal{P}, \mathcal{P}') \leq \min\{\mathcal{P}, \mathcal{P}'\}, \tag{9}$$

and the right side equality holds if and only if one of the partitions is a refinement of the other.

Consequently, the variation of information will be 0 if and only if the partitions are equal (up to permutations of indices of the parts), and will get bigger the more the partitions differ. It also satisfies the triangle inequality, so it is a metric in the space of clusterings of any given set.

4. Generating a random graph

The algorithm proposed here to generate a random graph which will serve as a null model is a modification of the switching algorithm described in [15]. Each step of this algorithm involves randomly selecting two edges AC and BD and replacing them with the new edges AD and BC (provided they didn't exist already). This leaves the degrees of each vertex A, B, C and D unchanged while shuffling the edges of the graph.

One way to adapt this algorithm to our weighted graphs (more specifically, complete weighted graphs, with weights in $[0, 1]$) is, given vertices A,B,C and D, transfer a certain weight $\bar{w}$ from w_{AC} to w_{AD}, and from w_{BD} to w_{BC}[6]. We will select only sets of vertices A, B, C, D such that $w_{AC} > w_{AD}$ and $w_{BD} > w_{BC}$, that is, we will be transfering weight from "heavy" edges to "weak" edges. For any value of $\bar{w}$, the weighted degree of the vertices remains constant, but if it is not chosen carefully there could have undesirable consequences.

[6] Here, w_{ij} refers to the weight of the edge between vertices i and j

4.1 Election of $\bar{w}$

Choosing large values of $\bar{w}$ could result in edge weights falling outside of the $[0, 1]$ interval in which all of our original values are contained, but small values will hardly have similarly small effects on the network. Restricting $\bar{w}$ to be as large as possible without edge weights falling out of $[0, 1]$, however, will favour a degenerate network in which most of the edge weights are either 0 or 1, which is also undesirable and unlike any network that could be obtained from correlations of time series.

If we bound the transfered weight to the difference between the strong and weak edges, the new weights will be upper and lower bounded by the initial strong and weak weights, respectively, which would avoid this issue entirely. In this case, the maximum transfered weight would have to be $\bar{w} = \min(w_{AC} - w_{AD}, w_{BD} - w_{BC})$. This results in one of the pairs of edges being exchanged, while in the other a certain weight equal or smaller than their difference is transfered. In this second case, it is important to note that the difference between the new edge weights will be smaller than the difference of the original weights (strictly smaller if $w_{AC} - w_{AD} \neq, w_{BD} - w_{BC}$).

The effect this has on the variance of the weights of the network can be seen on Figure 1. Unfortunately, as soon as the network starts to be significantly shuffled, the variance starts to fall. If we iterate the algorithm until the variation of information stops increasing, the variance has more than halved in our sample network.

As an alternative, we can impose the sample variance (given by $\frac{1}{n-1}\sum_{i,j=1}^{n}(w_ij - m)^2$, where m is the mean) to remain invariant after applying the transformation, and find the appropriate value of $\bar{w}$. The variance remains constant if and only if the following equality holds:

$$\begin{aligned} & (w_{AC} - m)^2 + (w_{BD} - m)^2 + (w_{AD} - m)^2 + (w_{BC} - m)^2 \\ = \quad & (w_{AC} - \bar{w} - m)^2 + (w_{BD} - \bar{w} - m)^2 + (w_{AD} + \bar{w} - m)^2 + (w_{BC} + \bar{w} - m)^2 \\ \iff \quad & 4\bar{w}^2 + 2\bar{w}(-(w_{AC} - m) - (w_{BD} - m) + (w_{AD} - m) + (w_{BC} - m) = 0 \\ \iff \quad & 2\bar{w}^2 + \bar{w}(-w_{AC} - w_{BD} + w_{AD} + w_{BC}) = 0. \end{aligned} \tag{10}$$

The solutions to this equation are $\bar{w} = 0$ (which is trivial and corresponds to not applying any transformation to the edge weights) and $\bar{w} = \frac{w_{AC}+w_{BD}-w_{AD}-w_{BC}}{2}$.

While this alternative can result in some weights falling outside of the interval $[0, 1]$, in the networks we studied it is very rare, so it is enough to discard these few steps to obtain the desired results.

Note that if all edge weights are either 0 or 1, in both cases this algorithm is equivalent to the original switching algorithm for discrete graphs, as in every step the transferred weight will be one if the switch can be made without creating double edges, or zero otherwise (which corresponds to the case in which the switch cannot be made).

4.2 Number of iterations

To determine how many iterations of the algorithm are enough to sufficiently "shuffle" the network, we study the variation of information of the resulting clustering respect to the initial one (Figure 1). As the algorithm transfers weight between the edges, the variation of information increases, until it stabilizes roughly after 10^4 iterations. Then, running 10^5 iterations to generate each random graph will be more than enough (there will be no improvement by iterating further) while still being very fast to compute. This is also consistent with the number of iterations found to be enough for the discrete case in [15].

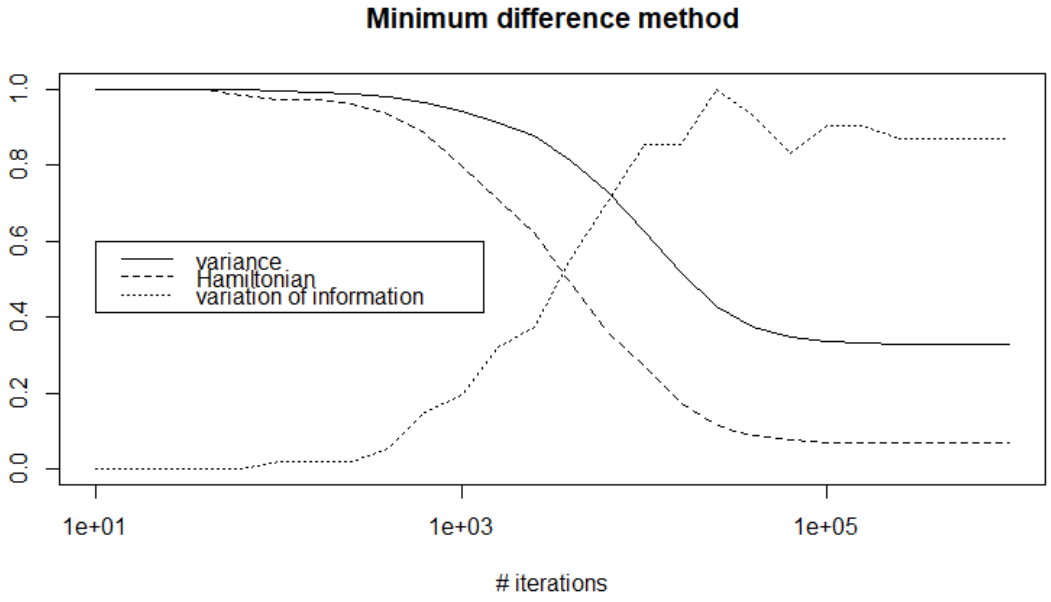


Figure 1: Normalized variance, Potts Hamiltonian and variation of information after applying the proposed algorithm with the minimum difference method. Horizontal axis is on logarithmic scale.

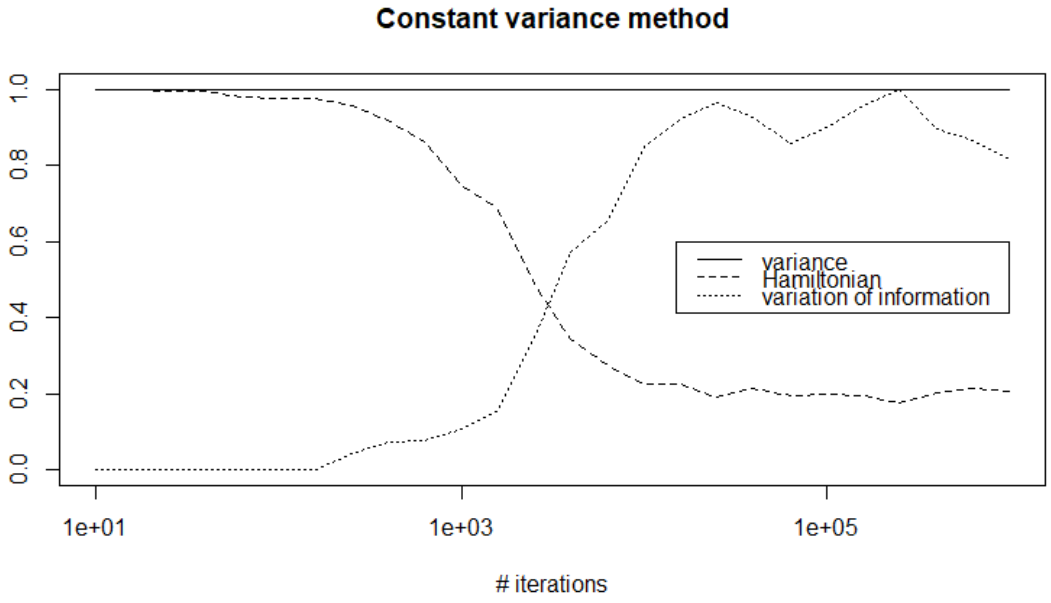


Figure 2: Normalized variance, Potts Hamiltonian and variation of information after applying the proposed algorithm with the constant variance method. Horizontal axis is on logarithmic scale.

5. Clustering validation

To check that the results given by the clustering algorithm when applied to our FX networks are significant, we generate a random network using the method described in Section 4 for every month in the 2009-2016 period. Ideally, we would expect to see that the clusters found in the real networks are much stronger than those in the randomized networks, which shouldn't have any meaningful community structure.

Figure 3: Number of appearances of communities of each size across the entire 2009-2016 period.

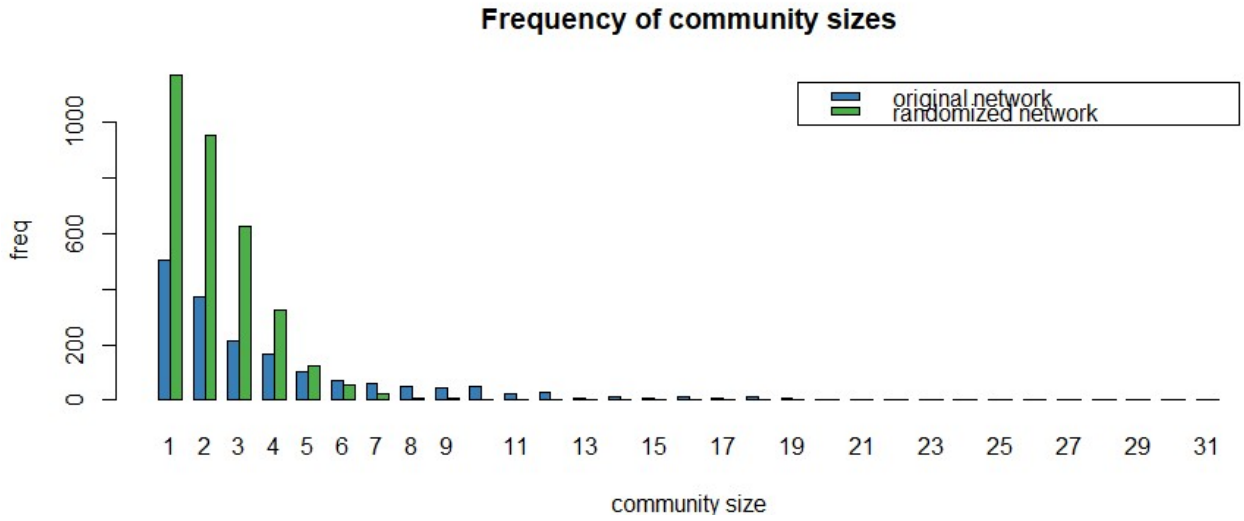
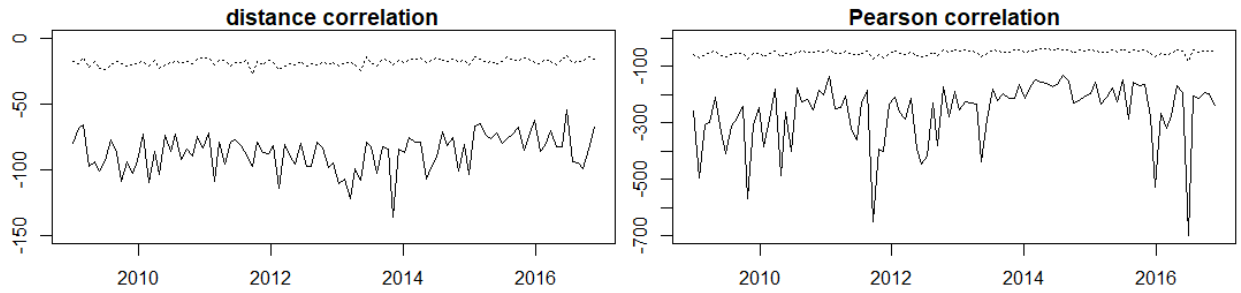


Figure 4: Hamiltonians for the original and randomized networks (solid and dashed lines, respectively), for both the distance and Pearson correlation methods.



In Figure 5 we can see that the clusterings of the randomized networks have many isolated vertices, and those that are grouped together are in smaller clusters than those we find in the original networks. In Figure 3 we verify that, across the entire observed period, the studied FX networks form larger communities than their randomized counterparts, and the number of nodes which are isolated or on very small communities is much smaller. Moreover, the value of the FX network Hamiltonian is consistently at least four times that of its corresponding randomized network using the distance correlation (Figure 4). With the Pearson

correlation, the Hamiltonian varies more but is also much lower than in its randomized network. Note though that the Hamiltonians cannot be compared across the different clustering methods, because with the Pearson correlation we need to take the inverses of each time series, resulting in a graph twice the order and four times the size.

Figures 6 and 7 show the evolution of the scoring functions introduced in Section 3. Even though we have considered several different types of scoring functions, in all cases the values of the original networks are better than those of their corresponding randomized ones[7]. Not only are the average scores better, but the results are consistent across all functions and periods of time. This, together with the much lower values achieved for the Hamiltonian, the objective function of the clustering algorithms, suggests that the observed community structure on our networks is significant and consistent.

While most of the values given by the scoring functions cannot be compared accross the two clustering methods due to differences in the networks (their size, for example), table 5 gives the percentage of increase of the real networks respect to the randomized models. We have observed the most dramatic increases on the internal connectivity based functions on the Pearson correlation networks (probably related to the inclusion of inverse exchange rates in the network), but the decreases in external connectivity (expansion, cut ratio) are better in the distance correlation networks. The distance correlation method also performs better with the clustering coefficient, whith an increase that almost doubles that of the Pearson correlation.

As for the improvements in the hamiltonian, the rates of increase for both methods are very similar, but the consistency observed by the distance correlation as opposed to the highs and lows observed over time with the Pearson correlation Hamiltonian (figure 4) could make it preferable.

Table 2: Means of the scoring functions over the 2009-2016 period for the randomized and observed networks, as well as the percentage of increase of the latter respect to the former.

	distance correlation			Pearson correlation		
	original	randomized	variation	original	randomized	variation
internal.density	0.83	0.81	1.90 %	0.85	0.91	-6.33%
edges.inside	24.02	3.00	701.49%	89.21	3.70	2313.02%
av.degree	4.44	1.62	174.37%	8.87	2.07	329.50%
FOMD	0.00	0.00	0.00%	0.00	0.00	0.00%
expansion	16.97	18.70	-9.24%	35.46	37.96	-6.57 %
cut.ratio.	0.23	0.25	-6.87%	0.24	0.25	-3.74%
conductance	0.89	0.95	-6.24%	0.87	0.95	-8.53%
norm.cut	0.91	0.97	-5.37%	0.89	0.96	-7.32%
max.ODF	0.94	0.97	-3.84%	0.92	0.97	-5.41%
average.ODF	0.94	0.97	-3.84%	0.92	0.97	-5.40%
flake.ODF	1.00	1.00	0.00%	1.00	1.00	0.00%
clustering.coef	0.89	0.75	17.82%	0.91	0.83	9.72 %
hamiltonian	-79.49	-18.24	335.79%	-259.35	-57.89	348.02 %

[7]Note that for some of the functions higher scores are desirable, while for others lower is better. Note also that most of the functions are normalized and take values in the interval $[0, 1]$, but the expansion and average degree are not.

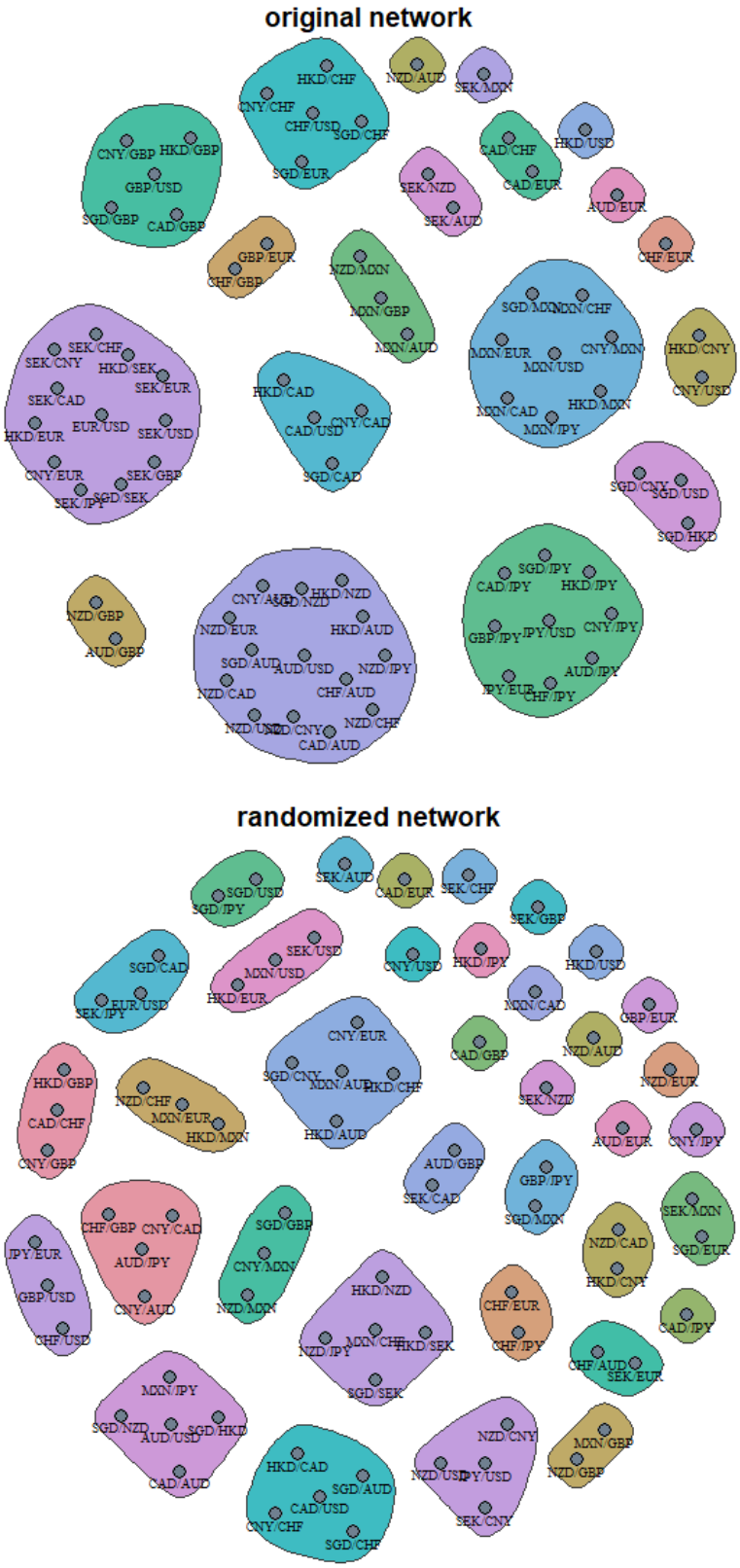


Figure 5: Plot of the obtained communities for the original and randomized networks of March 2009 using the distance correlation method.

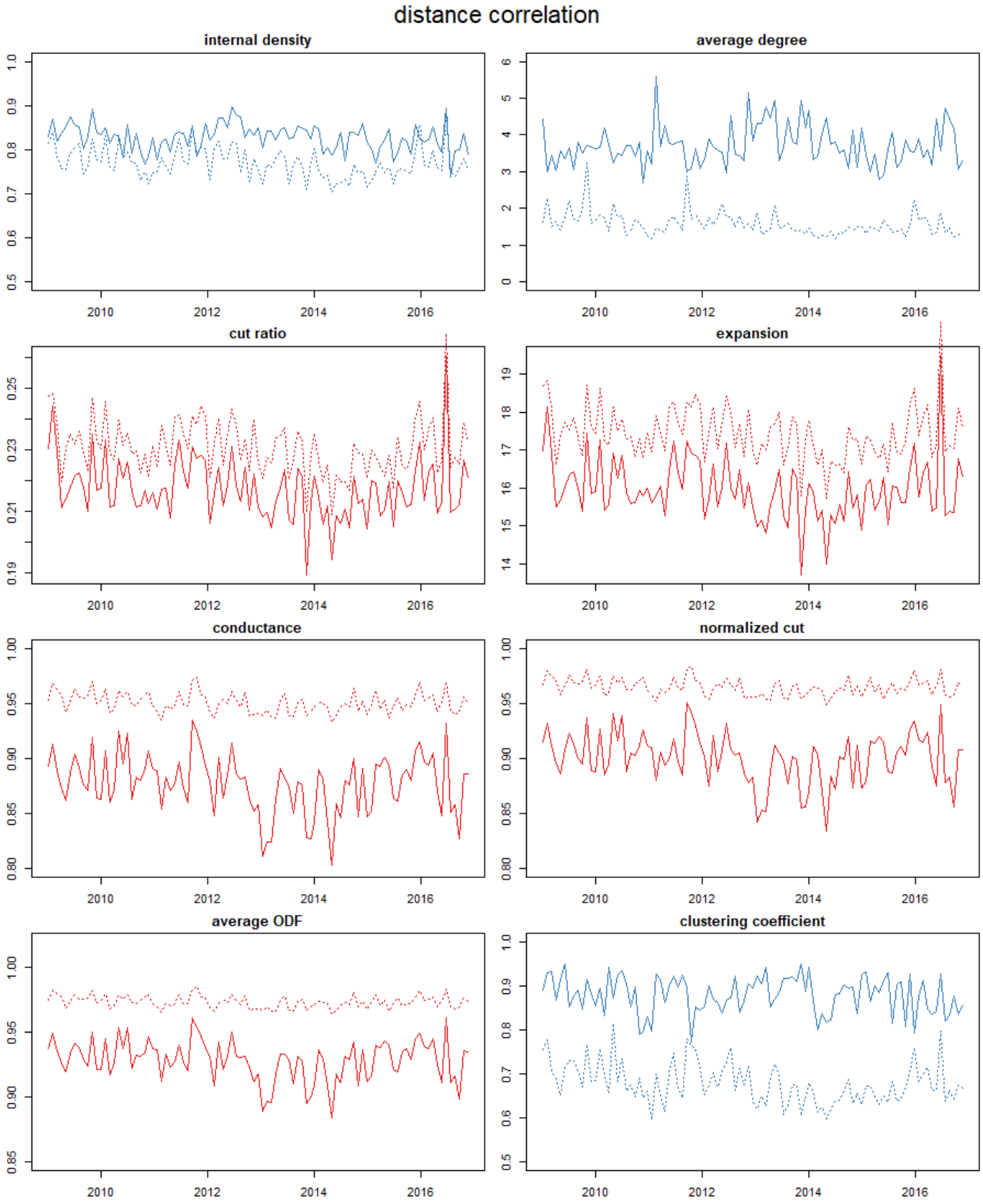


Figure 6: Scoring function values for the distance correlation networks. Solid and dashed lines correspond to the observed and randomized networks respectively. Scores in which higher values represent stronger clusters are represented in blue, while those in which lower is better are represented in red.

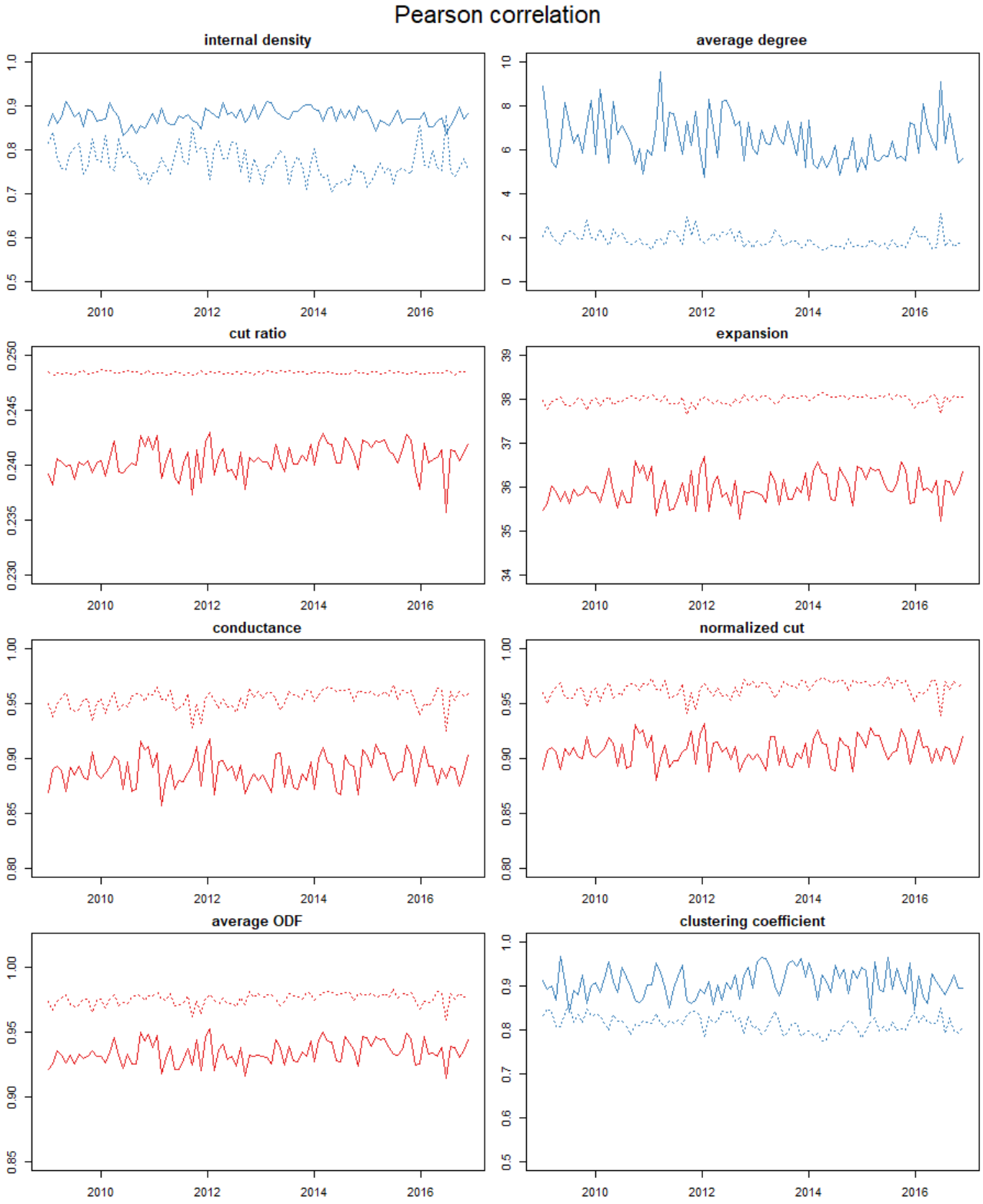


Figure 7: Scoring function values for the Pearson correlation networks. Solid and dashed lines correspond to the observed and randomized networks respectively. Scores in which higher values represent stronger clusters are represented in blue, while those in which lower is better are represented in red.

6. Cluster detection in simulated networks

The goal of this section is to generate artificial time series with an appropriate model such that some of them are correlated with each other. Applying the previously introduced methods will result in a network where the community structure is known, and which can be used to test clustering algorithms. If the correlations of time series belonging to the same communities are strong enough, we should expect the clustering methods to successfully and consistenly detect them.

6.1 VARMA model

Vector autoregressive moving average (VARMA) models are an extension of univariate ARMA models allowing modelling a vector X_t of stationary time series which may influence each other[16]:

Definition 6.1. X_t is a VARMA(p,q) process if it is stationary and if for every t,

$$X_t - \Phi_1 X_{t-1} - ... - \Phi_p Z_{t-p} = Z_t + \Theta_1 Z_{t-1} + ... + \Theta_q Z_{t-q}, \tag{11}$$

where Z_t is white noise

The coefficients of the Φ and Θ matrices will determine the dependencies between time series. For instance, setting them to diagonal matrices would be equivalent to having separate ARMA models for each of the series with the only dependencies being of each series with themselves. Alternatively, if they are block diagonal matrices (up to permutations of their indices, and all with the same block structure), there will be a community structure where each community will correspond to a block.

6.2 Simulation

The method presented in [12] is used to test the ability of our algorithms to detect existing clusters. Each step of the simulation involves generating four series following a VARMA(2,1) model with parameters:

$$\Phi_1 = \begin{pmatrix} 0.5 & 0.1 & 0 & 0 & 0 & 0 \\ 0.4 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0.1 & 0.6 & 0 \\ 0 & 0 & 0.3 & 0.3 & 0.4 & 0 \\ 0 & 0 & 0.4 & 0.5 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{pmatrix}, \Phi_2 = \begin{pmatrix} 0.3 & 0.1 & 0 & 0 & 0 & 0 \\ 0.5 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0.1 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.5 & 0.4 & 0 \\ 0 & 0 & 0.3 & 0.5 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.2 \end{pmatrix}, \tag{12}$$

$$\Theta_1 = \begin{pmatrix} 1 & \theta_1 & 0 & 0 & 0 & 0 \\ \theta_2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \theta_3 & \theta_4 & 0 \\ 0 & 0 & \theta_4 & 1 & \theta_5 & 0 \\ 0 & 0 & \theta_6 & \theta_7 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{13}$$

Given the block structure of the matrices, there is one cluster with series 1,2, another with series 3,4,5, and one with just series 6. The clustering algorithm is applied to check if it detects them. The simulation is run 1000 times counting the number of successes.

To check the performance of the algorithms at different correlation strengths, we set

$$\theta = (\theta_1, ..., \theta_7) = \alpha(0.08, 0.06, 0.07, 0.07, 0.09, 0.08, 0.08)$$

and run the simulation for several values of α. Table 3 contains the results for both the distance and Pearson correlations, with the number of successful attempts, the mean of the variation of information between the correct and the observed clusterings, and the number of clusterings with type I and type II errors[8].

Both methods have good rates of success, and in the cases where the correlation is strong they almost never miss (especially the distance correlation method). The algorithms only start giving questionable results when the values of θ are so low that the randmness of the simulation has a comparatively strong effect.

Table 3: Number of successes, mean of the variation of information respect to the correct clustering, and number of type I and II errors for both the distance and Pearson correlation clusterings.

	distance correlation				Pearson correlation			
α	# successes	VI mean	type I	type II	# successes	VI mean	type I	type II
1	392	0.3021	176	602	458	0.2461	141	492
2	774	0.1111	73	216	808	0.0800	59	148
5	941	0.0310	23	55	936	0.0294	35	37
10	997	0.0014	3	0	982	0.0092	18	0

7. Conclusions

The analysis of the FX networks with appropriate scoring functions allow us to conclude that they form a community structure not present in random networks. We have also verified that on networks where the existing community structure is known, the results given by the algorithms match the expected results.

As for the comparison between the distance and Pearson correlation methods, the results obtained here back up the validity of both, but the differences between them are small in most cases. The distance correlation does offer some improvements in the clustering coefficient, one of the most relevant scoring functions, and the values of its Hamiltonian achieved by the optimization algorithm, while similar on average to those of the Pearson correlation, are more consistent. Additionally, the fact that it runs the optimization algorithm on networks of half the number of nodes greatly reduces the computation time.

It is also worth noting that while this project was focused on financial networks, the methods proposed here are valid for evaluating the results of clustering algorithms on weighted networks in general.

[8] In this context, a type I error is made when two nodes are found to share a cluster and they don't in the theoretical and correct clustering (a false positive). Type II errors are made when two nodes are separated into different clusters while they shouldn't (false negatives). Note that there can be clusterings that contain both type I and type II errors.

References

[1] Daniel J. Fenn, Mason A. Porter, Peter J. Mucha, Mark McDonald, Stacy Williams, Neil F. Johnson, and Nick S. Jones. Dynamical clustering of exchange rates. *Quantitative Finance*, 12(10):1493–1520, 2012.

[2] R. N. Mantegna. Hierarchical structure in financial markets. *Eur. Phys. J. B.*, 11:193–197, 1999.

[3] Peter J Mucha, Thomas Richardson, Kevin Macon, Mason A Porter, and Jukka-Pekka Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *Science*, 328(5980):876–878, 2010.

[4] Edoardo Otranto. Clustering heteroskedastic time series by model-based procedures. *Computational Statistics & Data Analysis*, 52(10):4685–4698, 2008.

[5] J-P Onnela, Kimmo Kaski, and Janos Kertész. Clustering and information in correlation based financial networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):353–362, 2004.

[6] Martí Renedo and Argimiro Arratia. Clustering of exchange rates and their dynamics under different dependence measures. In *Proceedings of the First Workshop on MIning DAta for financial applicationS (MIDAS 2016) , Riva del Garda, Italy, September 19-23, 2016.*, pages 17–28, 2016.

[7] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.

[8] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[9] N. K. Bakirov G. J. Székely, M. L. Rizzo. Measuring and testing dependency by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 2007.

[10] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[11] Marina Meilă. Comparing clusterings - an information based distance. *Journal of Multivariate Analysis*, 98(5):873 – 895, 2007.

[12] Argimiro Arratia and Alejandra Cabaña. A graphical tool for describing the temporal evolution of clusters in financial stock markets. *Computational Economics*, 41(2):213–231, 2013.

[13] A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(11):3747–3752, 2004.

[14] Michael P. McAssey and Fetsje Bijma. A clustering coefficient for complete weighted networks. *Network Science*, 3(2):183–195, 2015.

[15] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *Arxiv preprint cond-mat/0312028*, 2003.

[16] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer New York, 2002.

[17] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[18] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[19] Maria L. Rizzo and Gabor J. Szekely. *energy: E-Statistics: Multivariate Inference via the Energy of Data*, 2016. R package version 1.7-0.

[20] Gilbert and P. D. *Brief User's Guide: Dynamic Systems Estimation*, 2006 or later.

[21] Quandl. Federal reserve economic data.

[22] Arno Fritsch. *mcclust: Process an MCMC Sample of Clusterings*, 2012. R package version 1.0.

A. R codes

The algorithms in this project were implemented in *R*[17] with the additional packages *igraph*[18], *energy*[19], *dse*[20], *Quandl*[21] and *mcclust*[22].

- `rewire.R`: Includes the implementation of the switching algorithm described in Section 4.
- `rewire_test.R` This script uses the `rewire` function to conduct the analysis and comparison of the original networks and their randomized counterparts found in Section 4.
- `scoring_functions.R`: Includes the implementation of the functions defined in Section 3.
- `modularity_functions`: Includes the implementation of the modularity and Potts Hamiltonian functions.
- `dcor_M.R` Given a vector of time series computes the matrix of correlations of every pair of series using the distance correlation metric.
- `Hmin_dyn_optimized.R`: Implementation of the minimization algorithm of the Potts Hamiltonian described in [8].
- `exchange_rates.R`: Contains the function `exchange.rates`, which given the start and end dates, as well as the desired frequency, downloads the data from Quandl[21] and returns it as a vector of time series.
- `plot_communities.R`: Using an auxiliary graph and manipulating edges appropriately, plots the network partitions avoiding community overlapping (see figure 5).

A.1 rewire.R

```
library(mcclust)

rewire <- function(M,itermax=10000,symmetric=TRUE, type="const_var",inverses=FALSE){
  #### M: the original adjacency matrix
  #### itermax: the number of iterations
  #### symmetric: if TRUE, each operation on an edge is repeated in its symmetric
      ↪ one (so M is kept symmetric)
  #### it should be set to TRUE for undirected graphs
  n <- dim(M)[1]
  variances <- array(dim=length(itermax)+1)
  variances[1]<-var(as.vector(M))

  if(inverses){
    M[1:(n/2),1:(n/2)] <- rewire(M[1:(n/2),1:(n/2)],symmetric=TRUE,type=type,
        ↪ inverses=FALSE)
    M[(n/2+1):n,(n/2+1):n] <- M[1:(n/2),1:(n/2)]
    M[(n/2+1):n,1:(n/2)] <- 1-M[1:(n/2),1:(n/2)]
    M[1:(n/2),(n/2+1):n] <- 1-M[1:(n/2),1:(n/2)]
```

```
    return(M)
  }

  for (i in 1:itermax){
    w <- sample(1:n,4,replace=FALSE) #sample without replacement to avoid self edges
        ↪  and duplicate edges
    wAC <- M[w[1],w[3]]
    wAD <- M[w[1],w[4]]
    wBC <- M[w[2],w[3]]
    wBD <- M[w[2],w[4]]


    if(wAC>wAD & wBD>wBC ){
      if (type=="const_var") {e <- (wAC+wBD-wAD-wBC)/2}
      else {e <- min(wAC-wAD,wBD-wBC)}

      if(type!="const_var" | (M[w[1],w[3]]-e>=0 & M[w[2],w[4]]-e>=0 & M[w[1],w[4]]+e
          ↪ <=1 & M[w[2],w[3]]+e<=1)){
        M[w[1],w[3]] <- M[w[1],w[3]] - e
        M[w[1],w[4]] <- M[w[1],w[4]] + e
        M[w[2],w[3]] <- M[w[2],w[3]] + e
        M[w[2],w[4]] <- M[w[2],w[4]] - e

        if (symmetric){ #repeat the switching operation on the symmetric edges
          M[w[3],w[1]] <- M[w[1],w[3]]
          M[w[4],w[1]] <- M[w[1],w[4]]
          M[w[3],w[2]] <- M[w[2],w[3]]
          M[w[4],w[2]] <- M[w[2],w[4]]
        }
      }
    }
    variances[i+1]<- var(as.vector(M))
  }
  return(M)
}

rewire_analysis <- function(M,from=10, to=1000000,by=0.20,type="const_var"){
  n_iter <- c(0,0,10^(seq(from=log10(from),to=log10(to),by=by)))
  d_iter <- diff(n_iter)
  n<- dim(M)[1]
  P <- sum(M)/(n*(n-1))
  c <- matrix(nrow=length(d_iter),ncol=n)
  d <- data.frame(n_iter=n_iter[2:length(n_iter)])

  for (i in 1:length(d_iter)){
    M <- rewire(M,itermax=d_iter[i],type=type)
```

```
    d$var[i]<-var(as.vector(M))
    g <- graph.adjacency(M, mode="undirected",weighted=TRUE,add.rownames=TRUE,diag=
        ↪ FALSE)
    c[i,] <- Hmin(g,P,gamma=gamma,itermax=10)
    d$hamiltonian[i] <- H.graph(g,P,c[i,],gamma)
    d$vi[i] <- vi.dist(c[1,],c[i,])
  }
  plot(d$var/max(d$var),type="l",ylim=c(0,1))
  lines(d$hamiltonian/max(d$hamiltonian),col="red")
  lines(d$vi/max(d$vi),col="blue")
  return(d)
}

plot_rewire_analysis <- function(d,legend_x=15000,legend_y=0.6,main="Constant␣
    ↪ variance␣method"){
  plot(d$n_iter,d$var/max(d$var),type="l",ylim=c(0,1),log="x",xlab="#␣iterations",
      ↪ ylab="",main=main)
  lines(d$n_iter,d$hamiltonian/max(d$hamiltonian),lty=2)
  lines(d$n_iter,d$vi/max(d$vi),lty=3)
  legend(legend_x,legend_y,c("variance","Hamiltonian","variation␣of␣information"),
      ↪ lty=1:3)
}

join_inverses <- function(v){
  n<-length(v)
  for (i in 1:(n/2)){
    m<-min(v[i],v[n/2+i])
    v[i]<-m
    v[n/2+i]<-m
  }
  return(v)
}
```

A.2 rewire_test.R

```
library(mcclust)
library(igraph)
library(Quandl)
library(energy)
library(linkcomm)

source("modularity_functions.R")
source("exchange_rates.R")
source("Hmin_dyn_optimized.R")
source("dcor_M.R")
```

```
source("plot_communities.R")
source("rewire.R")
source("scoring_functions.R")

#DATA SET CREATION
startdate <- as.Date("2009-01-01")
enddate <- as.Date("2016-12-31")
freq <- "daily"
seriestype <- "xts"
dyn.step <- 30 #interval of time for which changes in communities are observed

exchange.rates <- exchange.rates(startdate,enddate,freq,seriestype,reduced=FALSE)
returns <- diff(log(exchange.rates))[2:dim(exchange.rates)[1]]
n=dim(exchange.rates)[2]

startdate.dyn <- startdate
enddate.dyn <- as.Date(startdate)+dyn.step
i <- 1

dates <- seq(from=startdate, to=enddate, by=dyn.step)
c <- matrix(nrow=length(dates)-1,ncol=n) #c will contain the community of each
    ↪ vertex
c2 <- c


##############################
###CLUSTERING
##parameters
gamma=1.3

scores <- list()
scores_rewired <- list()

while (enddate.dyn<=enddate){
  print(paste("time␣step␣",i))
  Mcor <- dcor.M(returns[paste0(startdate.dyn,"/",enddate.dyn)]) ##AAQ: use
      ↪ distanceCor
  M2 <- rewire(Mcor,itermax=1e5)

  g <- graph.adjacency(Mcor, mode="undirected",weighted=TRUE,add.rownames=TRUE,diag=
      ↪ FALSE)
  g2 <- graph.adjacency(M2, mode="undirected",weighted=TRUE,add.rownames=TRUE,diag=
      ↪ FALSE)
  c[i,] <- Hmin(g,P,gamma=gamma,itermax=10)
  c2[i,] <- Hmin(g2,P,gamma=gamma,itermax=10)
  print(c("original␣H:",H.graph(g,P,c[i,],gamma)))
```

```
  print(c("rewired␣H:",H.graph(g2,P,c2[i,],gamma)))
  plot.com(g,c[i,],paste("from␣",dates[i],"␣to␣", dates[i+1]))
  plot.com(g,c2[i,],paste("from␣",dates[i],"␣to␣", dates[i+1]),sub="rewired")

  scores[[i]] <- scoring_functions(g,c[i,])
  scores_rewired[[i]] <- scoring_functions(g2,c2[i,])
  scores[[i]]["global","hamiltonian"] <- H.graph(g,P,c[i,],gamma)
  scores_rewired[[i]]["global","hamiltonian"] <- H.graph(g2,P,c2[i,],gamma)

  i<- i+1
  startdate.dyn <- enddate.dyn
  enddate.dyn <- as.Date(enddate.dyn)+dyn.step
}

##############################
### Analysis with scoring functions
global_scores <- function(L,dates=NULL){
  d <- data.frame(L[[1]]["global",])
  for (i in 2:length(L)){
    d[i,] <- L[[i]]["global",]
  }
  if (!is.null(dates)) row.names(d)<-dates[1:length(L)]
  return (d)
}

d <- global_scores(scores,dates = dates)
d_rewired <- global_scores(scores_rewired,dates=dates)
d["date"]<-row.names(d)


### PLOTS
x<-dates[1:dim(d)[1]]
col1 <- "#377eb8"; col2<-"#e41a1c" #we use col1 when "higher is better" and col2
    ↪ otherwise

par(mfrow = c(4, 2),mar = c(2,2,2,1),oma=c(0,0,2.5,0))
plot(x,d[,"internal.density"],type="l",ylim=c(.5,1),xlab="",ylab="",main="internal␣
    ↪ density",col=col1)
lines(x,d_rewired[,"internal.density"],lty=3,col=col1)
plot(x,d[,"av.degree"],type="l",ylim=c(0,6),xlab="",ylab="",main="average␣degree",
    ↪ col=col1)
lines(x,d_rewired[,"av.degree"],lty=3,col=col1)

plot(x,d[,"cut.ratio."],type="l",xlab="",ylab="",main="cut␣ratio",col=col2) #lower
    ↪ is better
lines(x,d_rewired[,"cut.ratio."],lty=3,col=col2)
```

```
plot(x,d[,"expansion"],type="l",xlab="",ylab="",main="expansion",col=col2) #lower is
    ↪  better
lines(x,d_rewired[,"expansion"],lty=3,col=col2)

plot(x,d[,"conductance"],type="l",xlab="",ylab="",main="conductance",ylim=c(.8,1),
    ↪ col=col2) #lower is better
lines(x,d_rewired[,"conductance"],lty=3,col=col2)

plot(x,d[,"norm.cut"],type="l",xlab="",ylab="",main="normalized␣cut",ylim=c(.8,1),
    ↪ col=col2) #lower is better
lines(x,d_rewired[,"norm.cut"],lty=3,col=col2)

plot(x,d[,"average.ODF"],type="l",xlab="",ylab="",main="average␣ODF",ylim=c
    ↪ (.85,1.02),col=col2) #lower is better
lines(x,d_rewired[,"average.ODF"],lty=3,col=col2)

plot(x,d[,"clustering.coef"],type="l",xlab="",ylab="",main="clustering␣coefficient",
    ↪ ylim=c(.5,1),col=col1) #higher is better
lines(x,d_rewired[,"clustering.coef"],lty=3,col=col1)

mtext("distance␣correlation", outer=TRUE, cex=1.4)
```

A.3 scoring_functions.R

```
library(igraph)

m_subgraph <- function(G,s){ #calculates the number of edges inside the subgraph
    ↪ induced by the edges in the list s
  ecount(subgraph(G,s))
}

m_subgraph_w <- function(G,s){ #weight of edges inside the subgraph induced by s
  sum(G[s,s])/2
}

degree_w <- function(G,v){ #weighted degree of vertex v
  sum(G[v,])
}

degree_s <- function(G,v,s){ #weighted degree of v over subgraph s (given by a list
    ↪ of vertices)
  sum(G[v,s])
}
```

```
median_degree_w <- function(G){
  M <- get.adjacency(g,attr="weight")
  median(apply(M,1,sum)) #computes the median of the degree sequence
}

cs_w <- function(G,s){ #sum of weight of edges connecting vertices of s to the rest
    ↪ of the graph
  sum(g[s,-s])/2 #sum the elements of rows in s and columns not in s (divide by two
      ↪ because each edge is counted twice)
}


#network metrics
internal_density_s <- function(G,s){
  n <- length(s)
  if (n<=1) return(0)
  else return(2*m_subgraph_w(G,s)/(n*(n-1)))
}

average_degree <- function(G,s){
  2*m_subgraph_w(G,s)/length(s)
}

frac_over_median <- function(G,s,dm="default"){
  if(dm=="default") dm <- median_degree_w(G) #if median degree has not been given,
      ↪ compute it
  if (length(s)>=2)deg.seq <- apply(G[s,s],1,sum)
  else deg.seq <- 0
  sum(deg.seq>dm)/length(s)
}

expansion <- function(G,s){
  cs_w(G,s)/length(s)
}

cut_ratio <- function(G,s){
  n <- length(s)
  cs_w(G,s)/(n*(gorder(G)-n))
}

conductance <- function(G,s){
  cs_w(G,s)/(2*m_subgraph_w(G,s)+cs_w(G,s))
}

normalized_cut <- function(G,s){
  cs <- cs_w(G,s)
```

```
  ms <- m_subgraph_w(G,s)
  m <- sum(M <- get.adjacency(g,attr="weight"))/2
  cs/(2*ms+cs)+cs/(2*(m-ms)+cs)
}

max_odf <- function(G,s){
  first <- TRUE
  M <- 0
  for (i in s){
    out_degree <- sum(G[i,-s]) #out degree (sum of edges leaving s) of vertex i
    odf <- out_degree/degree_w(G,i)
    if (first|odf>M) M <- odf
  }
  return(M)
}

av_odf <- function(G,s){
  sum_odf <- 0
  for (i in s){
    out_degree <- sum(G[i,-s]) #out degree (sum of edges leaving s) of vertex i
    odf <- out_degree/degree_w(G,i)
    sum_odf <- sum_odf + odf
  }
  return(sum_odf/length(s))
}

flake_odf <- function(G,s){
  x <- 0
  for (i in s){ #x will count how many vertices of s have smaller edge weight sum
      ↪ pointing inside than outside
    if (sum(G[i,s]<sum(G[i,-s]))) x <- x+1
  }
  return(x/length(s))
}

clustering_coef_w <- function(G,n_step=100){
  average <-0
  A <- get.adjacency(G,attr="weight")
  for (i in 1:n_step){
    t <- i/n_step
    At <- A>=t
    Gt <- graph.adjacency(At,mode="undirected") #unweighted graphs with edges where
        ↪ the weight is above the threshold t
    if (ecount(Gt)>1) {
      trans <- transitivity(Gt) #if there are no edge we consider the transitivity
          ↪ to be 0 (but igraphs transitivity function would return NaN)
```

```
      if (trans!="NaN")average <- average+ trans
    }
  }
  return(average/n_step)
}

clustering_coef_subgraph <- function(G,s,n_step=100){
  Gs <- induced_subgraph(G,s)
  clustering_coef_w(Gs,n_step)
}

###################
#other
relabel <- function(c){
  c2 <- c
  j <- 1
  for (i in unique(c)){
    c2[c==i] <- j
    j <- j+1
  }
  return(c2)
}


##################
scoring_functions <- function(G,com,fix_labels=TRUE){
  if (fix_labels) com <- relabel(com)
  n_com <- max(com)
  function_names <- c("internal␣density","edges␣inside","av␣degree","FOMD","
      ↪ expansion",
                      "cut␣ratio,","conductance", "norm␣cut", "max␣ODF","average␣ODF",
                          ↪ "flake␣ODF","clustering␣coef","modularity")
  D <- data.frame(matrix(nrow=n_com+1,ncol=length(function_names)))
  colnames(D) <- function_names
  rownames(D) <- c(1:n_com,"global")
  for (i in 1:n_com){
    s <- which(com==i) #s contains the indices of vertices belonging to cluster i
    D[i,]<- c(internal_density_s(G,s),m_subgraph_w(G,s),average_degree(G,s),frac_
        ↪ over_median(G,s),expansion(G,s),
              cut_ratio(G,s),conductance(G,s),normalized_cut(G,s),max_odf(G,s),av_odf
                  ↪ (G,s),flake_odf(G,s),
              clustering_coef_subgraph(G,s),0)
  }
  D[n_com+1,]<-colMeans(D[1:n_com,])
  D["modularity"]<-"-"
  D[n_com+1,"modularity"] <- modularity(G,com,weight=E(G)$weight)
```

```
  D[n_com+1,"clustering␣coef"]<- sum(D[1:n_com,"clustering␣coef"])/ sum(D[1:n_com,"
      ↪ clustering␣coef"]>0) #mean of non-zero elements
  D[n_com+1,"internal␣density"]<- sum(D[1:n_com,"internal␣density"])/ sum(D[1:n_com,
      ↪ "internal␣density"]>0)
  D[n_com+1,"edges␣inside"]<- sum(D[1:n_com,"edges␣inside"])/ sum(D[1:n_com,"edges␣
      ↪ inside"]>0)
  D[n_com+1,"av␣degree"]<- sum(D[1:n_com,"av␣degree"])/ sum(D[1:n_com,"av␣degree"
      ↪ ]>0)
  return(D)
}
```

A.4 modularity_functions.R

```
#returns the matrix with the images of the kronecker delta function of c[i],c[j]
delta <- function(c){
  k<-length(c)
  d <- diag(k)
  for (i in 1:k){
    for (j in 1:i){
      if(c[i]==c[j]) {
        d[i,j] <- 1
        d[j,i] <- 1
      }
    }
  }
  return (d)
}

#Modularity function defined in (5)
Q <- function(A,P,c){
  q <- sum((A-P)*delta(c))
  return(q/sum(A))
}


#Hamiltonian of Potts spin glass from adjacency matrix A. For gamma=1, H=2*m*Q
H.matrix <- function(A,P,c,gamma=1){
  X <- (A-gamma*P)
  X <- data.matrix(X)
  diag(X) <- 0
  h <- sum(X*delta(c))
  return (h)
}
```

```
H.graph <- function(g,P,c,gamma=1){
  #Hamiltonian of Potts spin glass.
  # g: input graph
  # P: edge strength of null model. In this network P is constant.
  # c: vector that for each vertex gives its community
  # gamma: parameter. For gamma=1, H is proportional to the modularity function.

  return(H.matrix(get.adjacency(g,attr="weight"),P,c,gamma))
}
```

A.5 VARMAtest.R

```
library(dse)

##array of dim axpxp = a pxp-matrices:
##Below read c(1,.5,.3,...) as entry [1,1] of matrix 1,2,3; and so on by row and
    ↪ then columns: entry [1,2] of matrix 1,2,3 ...
AR=array(c(1,.5,.3,0,.1,.1,0,0,0,0,0,0,0,0,0,0,0,0,
          0,.4,.5,1,.5,.4,0,0,0,0,0,0,0,0,0,0,0,0,
          0,0,0,0,0,0,1,.2,.3,0,.1,.1,0,.6,.3,0,0,0,
          0,0,0,0,0,0,0,.3,.3,1,.3,.5,0,.4,.4,0,0,0,
          0,0,0,0,0,0,0,.4,.3,0,.5,.5,1,.3,.3,0,0,0,
          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,.1,.2),c(3,6,6))
##select theta parameters
#t1=0.06; t2=0.06; t3=0.07; t4=0.07;
alpha <- 10
t <- alpha*c(.08,.06,.07,.07,.09,.08,.08)
B24=array(c(1,t1,0,0, t2,1,0,0,0,0,1,t3,0,0,t4,1),c(4,4))
B24=array(c(1,t[1],0,0,0,0,
           t[2],1,0,0,0,0,
           0,0,1,t[3],t[4],0,
           0,0,t[4],1,t[5],0,
           0,0,t[6],t[7],1,0,
           0,0,0,0,0,1),c(6,6))

var2=ARMA(A=AR,B=B24) ##multivariate ARMA(2,1) = VARMA(2,1)
length=30 ##length of simulated series
I= 1000 ##total number of iterations=repeated experiments

correct=0
type1=0 #an error is type1 when 2 nodes share a cluster and shouldn't
type2=0 #an error is type2 when 2 nodes should share a cluster and don't

cluster_table<-function(c){ #returns a matrix of "cluster adjacencies" (M[i,j]=1 iff
    ↪ i,j are in the same cluster)
```

```
  M<-matrix(nrow=length(c),ncol=length(c))
  for(i in 1:length(c)){
    M[i,]<- (c[i]==c)
  }
  return(M)
}

correct_cluster <- c(1,1,2,2,2,3)
table1<-cluster_table(correct_cluster)
varinf <- array(dim=I)
for(i in 1:I){
  simdata <- simulate(var2,sampleT=length)

  Mcor <- dcor.M(simdata$output)
  #Mcor <- cor(simdata$output,method="pearson")
  #Mcor <- 0.5*(Mcor+1)

  n <- dim(Mcor)[1]
  Mcor <- Mcor-diag(n)
  P <- (sum(Mcor))/(n*(n-1))
  g <- graph.adjacency(Mcor, mode="undirected",weighted=TRUE,add.rownames=TRUE,diag=
      ↪ FALSE)
  cluster2 <- Hmin(g,P,gamma=1.1,itermax=10)
  table2 <- cluster_table(cluster2)
  print(cluster2)

  if (sum(table2-(table2&table1))>0) type1 <- type1+1
  if (sum(table1-(table2&table1))>0) type2 <- type2+1

  varinf[i] <- vi.dist(correct_cluster,cluster2)
  if (varinf[i]<1e-10) correct = correct+1;
  }
print(c("correct:␣",correct))
print(c("type␣1:␣", type1))
print(c("type␣2:␣", type2))
print(c("VI␣mean:␣",mean(varinf)))
```

A.6 plot_communities.R

```
#Creates a new graph with an appropiate layout to represent the communities without
    ↪ overlapping
library(colorspace)

plot.com <-function(g,c,main="",sub=""){
  n <- length(V(g))
```

```
  Mplot <- matrix(ncol=n,nrow=n,0)
  colnames(Mplot)<-labels(V(g))
  rownames(Mplot)<-labels(V(g))
  for(j in 1:n){
    b <- TRUE
    for(k in 1:n){
      if (c[j]==c[k] & b){
        Mplot[j,k] <- 1
        b <- FALSE
      }
    }
  }

  gplot <- graph.adjacency(Mplot, mode="undirected",weighted=TRUE,add.rownames=TRUE,
      ↪ diag=FALSE)
  l <- layout.fruchterman.reingold(gplot)
  groups <- list()
  j <- 1 #index of the community
  for(i in c){
    groups[[j]] <- which(c==i) #adds an element to the list with the indices of
        ↪ nodes in community i
    j<-j+1
  }
  lab <- labels(V(gplot))
  for (i in 1:n) lab[i] <- paste0(substr(lab[i],1,3),"/",substr(lab[i],4,6))
  V(gplot)$name <- lab
  colours <- rainbow_hcl(length(groups),start = -1000, end = 300)
  plot(gplot,layout=l, vertex.size=4, vertex.label.dist=0.15,vertex.label.degree=pi/
      ↪ 2,vertex.label.cex=0.55,
       edge.lty="blank",main=main,mark.groups=groups,mark.col=colours,mark.border="
           ↪ #383838" ,vertex.color="slategray",
       vertex.label.color="black",mark.shape=1/2,mark.expand=17,sub=sub)
}
```