

Ecography

ECOG-00506

Marquitti, F. M. D., Guimarães Jr, P. R., Pires, M. M. and Bittencourt, L. F. 2014. MODULAR: software for the autonomous computation of modularity in large network sets. – *Ecography* 37: xxx–xxx.

Supplementary material

Supplementary Material

Appendix 1: Metrics

The metrics used in MODULAR are the Newman and Girvan (2004) metric and its modified version for bipartite networks, which was proposed by Barber (2007). The first metric is calculated as follows:

$$Q = \sum_{i=1}^{N_M} \left[\frac{E_i}{E} - \left(\frac{k_i}{2E} \right)^2 \right], \quad (1)$$

where N_M is the number of modules, E_i is the number of links in module i , E is the number of links in the complete network, and k_i is the sum of the degrees of the nodes within module i . Thus, a good partition has many links inside the modules and as few links as possible between the modules.

The bipartite version of the metric is calculated as follows:

$$Q_B = \sum_{i=1}^{N_M} \left[\frac{E_i}{E} - \left(\frac{k_i^C \cdot k_i^R}{E^2} \right) \right], \quad (2)$$

where k_i^C is the sum of the degrees of the nodes within module i that belong to set C , and k_i^R is the sum of the degrees of the nodes within module i that belong to set R . The equation for Q_B differs from the equation for Q because in the second term, only edges between nodes of different sets are considered.

Appendix 2: Language, Libraries, and Algorithms

We implemented the MODULAR software in *C*. Our code depends on two libraries: *igraph* and the GNU Scientific Library (GSL). The *igraph* library provides data structures for graph representation, manipulation functions and functions for community structure detection. The GSL library is mainly used to generate the random numbers that are needed by the optimization methods. By implementing a set of functions utilizing those two libraries, we were able to develop an easy-to-use software program for the detection of modules with maximum modularity. The *igraph* library utilized is the 0.6 version, which supports UCINET's DL file format. We used the *igraph_read_graph_dl* function to read the network into a graph data structure. In addition, we used *igraph* data structures called *membership*

vectors to represent the modules of the network. The SA (Kirkpatrick et al. 1983) method was implemented using a data structure from GSL that contains a number of running parameters, such as the initial temperature and its damping factor. The *fast greedy* (Clauset et al. 2004) and the *spectral partitioning* (Newman 2006) methods are functions called from the igraph library. These two methods are faster than simulated annealing, but their searches are based on local decisions and, thus, eventually lead to lower modularities than those found with SA. In MODULAR, we combined the speed of these two methods to find a reasonable solution with the broader search that is provided by the simulated annealing method by creating two hybrid methods. The first hybrid approach (*Hyb-SP*) first runs the SP method, and its module configuration output is utilized as the input for the SA method using a reduced initial temperature. The same technique is applied in the second hybrid method, except that the FG method is used instead of the SP method (*Hyb-FG*).

Appendix 3: MODULAR Functioning

The basic MODULAR functioning is represented in Figure A1. The set of input files can include two kinds of files: networks represented as matrices in text files (extension *.txt*) or lists of interactions, such as those used by the UCINET program (extension *.dl*) (Borgatti et al. 2002). Both kinds of input files can contain representations of *bipartite networks* or *unipartite networks*. If the input data are text files containing matrices, each *.txt* file is a binary matrix in which the columns are separated by tabulation or a space. The user must not include any row or column labels in the input file. The representation of unipartite networks should contain square matrices in which the rows i and columns i depict the same element in the $N \times N$ matrix. These matrices are symmetrical, and the program will read only the input data from the upper triangular part.

The current version of the program is not intended to be used with directed networks. If the matrices in the text files represent bipartite $N \times M$ networks, the rows and columns represent different sets of elements, and edges exist only between elements of the two different sets. For example, the rows can represent islands, and the columns can represent species in a network describing the species co-occurrence across islands (Carstensen et al. 2012). In contrast, if the input data are UCINET files, the representation of unipartite networks should follow the UCINET file formats with the *format* field set to contain the format type *edgelist1* followed by an edge list of the node pairs. The representation of bipartite networks should set the *format*

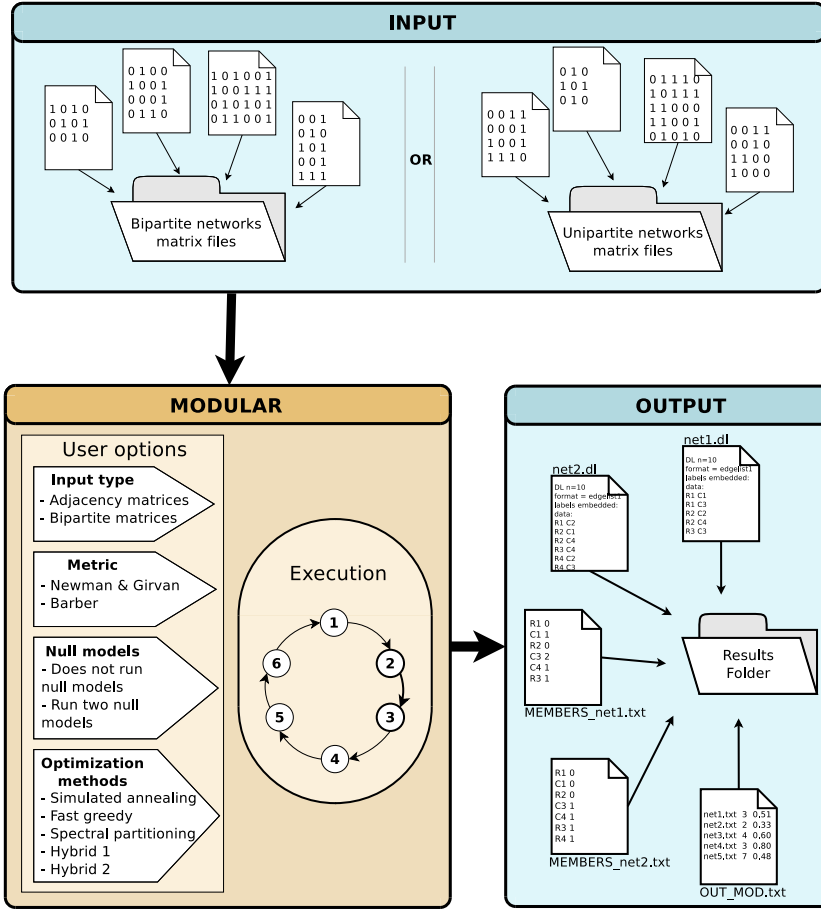


FIG. A1. **Design of the MODULAR software.** The program design can be divided into three steps. The first step is the *input* file preparation. The user may choose between representations of bipartite or unipartite networks. The second step includes the running of **MODULAR**. After the user answers some questions, MODULAR will execute six steps (see the execution explanation in the text). Finally, MODULAR will produce *output* files with the results.

field to *edgelist2*, followed by an edge list of the node pairs. The first node of the pair is an element of one set, and the second node is an element of the other set. Therefore, the first option that MODULAR gives the user is the selection of different types of input files that represent bipartite and unipartite networks.

The second option given to the user regards the modularity metric: Newman and Girvan's - Q (Newman and Girvan 2004) or Barber's modularities - Q_B (Barber 2007) (see Appendix 1 for details). If the user chooses the traditional Q metric, there are five optimization algorithms that can be used: (i)

fast greedy (FG) (Clauset et al. 2004, Wakita and Tsurumi 2007), (ii) simulated annealing (SA) (Guimerà and Amaral 2005), (iii) spectral partitioning (SP) (Newman 2006), (iv) a hybrid of simulated annealing and spectral partitioning (Hyb-SP), and (v) a hybrid of simulated annealing and fast greedy (Hyb-FG). The optimization algorithms differ in the method used to search the network partition that maximizes the modularity measurement (objective function).

The FG method searches the maximum modularity by only accepting partitions that exhibit a higher modularity than the previous partition. It is a greedy optimization method, proposed by Clauset *et. al* (2004) for the analysis of modularity and improved by Wakita & Tsurumi (2007). The method is summarized below:

FG optimization:

- 1) Compute the modularity of the current partition of the network.
- 2) Merge communities and accept the new partition that provides the highest modularity.
- 3) Stop when no partition merge can increment the modularity.

The SA method searches for the maximum modularity while trying to avoid getting stuck at local maxima (Kirkpatrick et al. 1983). This time-consuming method first partitions the network into modules with one node each, thus generating an initial configuration of N modules of size 1. The method then randomly moves nodes between modules and computes the resulting modularity of the new module configuration. If the new modularity is higher than that obtained with the previous module configuration, the algorithm takes the new module configuration as the current solution. On the other hand, if the new modularity is lower than that obtained with the previous module configuration, the algorithm accepts it as the new solution with a given probability (Metropolis et al. 1953), thus simulating a system of particles in which the energy levels follow a Boltzmann distribution. Using this technique, the algorithm avoids getting trapped at local maxima by moving through sub-optimal solutions and potentially converging to a maximal point (Kirkpatrick et al. 1983). This avoidance is possible because of the *temperature* parameter, which is directly linked to the probability of accepting new solutions. The temperature decreases gradually during the process, and this decay is governed by the parameter *cooling factor* parameter. The initial temperature and the cooling factor, both set by the user, determine the static annealing schedule. At each step, the temperature is divided by the

cooling factor, thereby decreasing the probability to accept worse solutions (lower modularity values) than the current one. The steps are repeated until the SA temperature reaches a threshold or the algorithm undergoes a predetermined number of iterations without any changes in the solution. Higher initial temperatures, higher number of iterations, and smaller cooling factors will usually lead to best estimates of modularity but will increase running time. If the user chooses to maximize Barber’s metric, the simulated annealing method is the only search method available in the current version of the program. The method is summarized below:

SA optimization:

- 1) Compute the modularity in the current partition of the network.
- 2) Generate a new module partitioning by randomly exchanging nodes and merging/splitting modules, compute the modules, and determine the modularity.
- 3) Accept the new partition if the modularity increases. If the modularity decreases, accept the new partition with a given probability $P(t)$ based on the difference between the new and the previous modularity values (Δ Energy) and according to the temperature ($T(t)$) used to run the SA, such as $P(t) = e^{-\Delta Energy/T(t)}$, where $T(t) = \frac{T(t-1)}{c}$, with the initial temperature $T(0)$ and cooling factor c set by the user.
- 4) Stop if the solution does not change after a predetermined number of iterations or when the minimum temperature chosen to run the SA is reached.

The SP method is based on matrix spectra. This fast method divides the network into sequential groups according to the entries in the leading eigenvector (Clauset et al. 2004).

SP optimization:

- 1) Establish the modularity matrix (matrix B, see Newman 2006).
- 2) Divide the network into two groups according to the entries in the leading eigenvector.
- 3) Repeat the process until none of the sub-matrices have a positive eigenvalue.

In MODULAR, we introduced two additional hybrid methods that combine the speed of the SP and FG approaches with the larger search space of simulated annealing. In the hybrid approaches, the SA input is the output of the faster method (SP or FG). Consequently, the optimization can achieve better results. After these steps, the user can choose to verify the modularity found by the maximization against two different null models.

Based on the options above, MODULAR iteratively executes up to six steps (“Execution”, in Figure A1) for each *.txt* or *.dl* file present in the matrices folder specified by the user:

For each *.txt* or *.dl* file in the specified folder, do:

- 1) Read the matrix A or the edge list from the input file according to the specified type (representation of a bipartite or unipartite network);
- 2) If the input file is a matrix, remove all null rows and null columns present in the matrix to generate a new matrix A' ;
- 3) If the input file is a matrix, generate a *.dl* UCINET (Borgatti et al. 2002) file with the network represented by matrix A' ;
- 4) Read the *.dl* file and load the network into the computer memory using data structures that can be manipulated by the *igraph* library functions;
- 5) Maximize the modularity of the network by running the method selected by the user (SA, SP, FG, Hyb-SP, or Hyb-FG);
- 6) Generate the theoretical networks according to the null model (if requested by the user), and run step 5 for each generated network.

A set of output files is generated after these steps. There are three main output files. If the input files are named *MatrixName.txt*, MODULAR generates a file with the list of interactions named *MatrixName.dl* in the UCINET edgelist1 format. The first line indicates the number of nodes in the network. If MODULAR was used to analyze bipartite network representations, the rows will be named R followed by a number (for example, $R2$ for the second row of the input file), and the columns will be named C followed by a number (for example, $C3$ for the third column of the input file). If the data set is composed of unipartite network representations, the lines will be named L followed by a number (for example, $L2$ for the second line of the input file). In both cases, the number refers to the order found in the input file.

A set of output files is generated after these steps. The main output is the *OUT_MOD.txt* file. This file contains, for each input file, the value of the modularity metric and the number of modules found by the metric and algorithm chosen. If the user chooses to run the null models, the *OUT_MOD.txt* file will also contain the proportion of null matrices with a modularity higher than that in the input file. An output file named *MEMBERS_MatrixName.txt* is generated for each input file named *MatrixName.txt*. This file has two columns: the first column has the labels of the matrix lines from the *.dl* file, and the second column indicates the module to which each line belongs. If null model analysis is performed, there will be two additional outputs: *OUT_1_MatrixName.txt*, which includes the modularities of Erdős-Rényi model, and *OUT_2_MatrixName.txt*, which includes the modularities of “null model 2”. Both files have as many lines as the number of null model runs chosen by the user.

Appendix 4: Performance test

To explore the performance of the different optimization algorithms, we used ecological networks available at the Interaction Web Database website (<http://www.nceas.ucsb.edu/interactionweb>). We used MODULAR to compute the modularity of 51 bipartite ecological networks (representing host-parasite, ant-plant, plant-herbivore, plant-pollinator, and plant-seed disperser interactions) and 27 unipartite ecological networks (food webs), and we registered the running time and the modularity value estimated using each algorithm.

Figure A2 shows the modularity values, Q , of bipartite (panel A) and unipartite (panel B) reached by each algorithm in relation to the value reached by the SA optimization. Note that the hybrid algorithms often reach similar values of modularity when compared to the SA algorithm (dashed line), whereas the SP and FG algorithms may underestimate Q .

Figure A3 shows the differences in the running time of each optimization algorithm. As expected, the running time increases with network size, i.e., number of species, but the FG and SP algorithms are generally faster. The linear regression to log-log values of the three slower optimization algorithms (SA, Hyb-SP, and Hyb-FG) show that the linear regression for the SA algorithm does not result in the largest slope, but its intercept is higher than the two other algorithms. The intercept and slope to bipartite networks were 0.305 and 0.958, respectively, for SA; -0.048 and 0.981 for Hyb-SP; and -0.112 and 1.000 for Hyb-FG. Thus, these performance tests suggest that, for bipartite networks, the SA algorithm can run faster than the hybrid algorithms

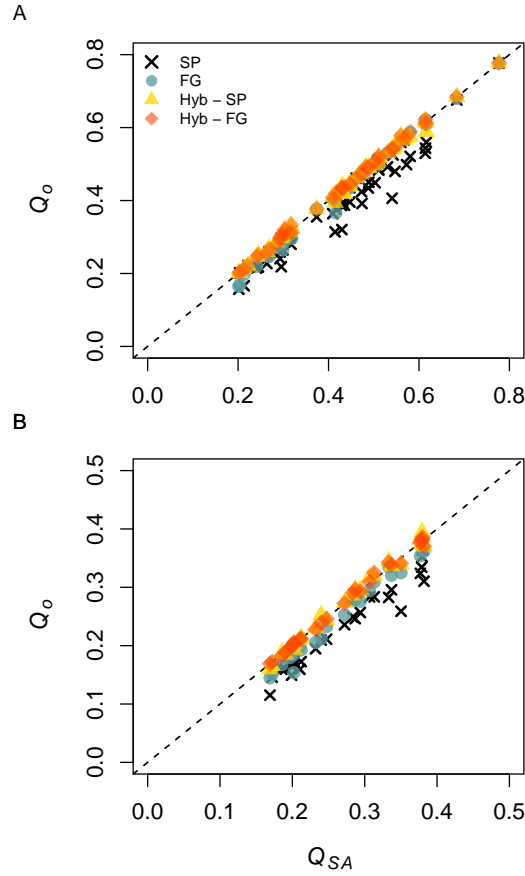


FIG. A2. **The modularity of each network obtained using different optimization methods (Q_o) in relation to the modularity computed using the simulated annealing algorithm (SA).** SP: spectral partitioning, FG: fast greedy, Hyb-SP: hybrid of spectral partitioning and simulated annealing, Hyb-FG: hybrid of fast greedy and simulated annealing. Points below the dashed line mean the estimated value of Q for that network using that particular optimization method was smaller than the value obtained using the SA algorithm. A) Results for bipartite and B) for unipartite ecological networks.

only when the network is larger than 10^{15} possible edges.

The intercept and slope for unipartite networks were -1.486 and 1.372, respectively, for SA; -1.514 and 1.300 for Hyb-SP; and -1.847 and 1.399 for Hyb-FG. With food webs, the hybrid version of Hyb-SP was always faster than the SA algorithm. The SA optimization algorithm can run faster than Hyb-FG only when the network has more than 10^{13} possible edges. In contrast, FG is clearly the fastest algorithm for both bipartite and unipartite networks, including very large ecological networks. With these results, we show that MODULAR is suitable for a wide range of networks, presenting

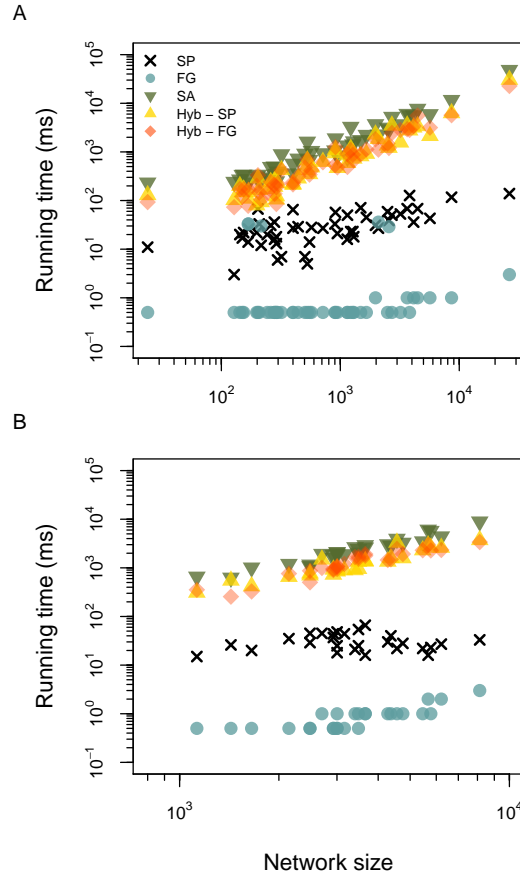


FIG. A3. **Running time of each optimization method as a function of network size.** SP: spectral partitioning, FG: fast greedy, SA: simulated annealing, Hyb-SP: hybrid of spectral partitioning and simulated annealing, Hyb-FG: hybrid of fast greedy and simulated annealing. Network size measured by the number of possible edges: RC for bipartite networks; and $N(N-1)/2$ for unipartite networks. Graph on log-log scale. A) Results for bipartite and B) for unipartite ecological networks.

options to the user to choose the best combination to save time and obtain reasonable estimates of modularity. We also show that hybrid algorithms usually perform faster than SA.

Appendix 5: Null models

The null models implemented in MODULAR are the Erdős-Rényi model (Erdős and Rényi 1959) and “null model 2” (Bascompte et al. 2003). In the Erdős-Rényi model, each pair of nodes has the same probability of being

connected by an edge. Thus, the probability of a pair (i, j) to be linked is given by

$$P(i, j) = \frac{E}{R \cdot C}, \quad (3)$$

where E is the number of edges in the network. If the network is bipartite, R and C are the number of nodes in each set of the network. For unipartite networks, $R = C$ and the probability of a pair (i, j) to be linked is given by

$$P(i, j) = \frac{2E}{R \cdot (R - 1)}. \quad (4)$$

In “null model 2” (Bascompte et al. 2003), the probability of a pair being connected by an edge is proportional to the number of edges that the nodes have. Thus, the probability of a pair (i, j) to be linked is given by

$$P(i, j) = \frac{1}{2} \left(\frac{k_{i \in R}}{C} + \frac{k_{j \in C}}{R} \right), \quad (5)$$

where k_i is the number of edges of node i in the R partition, and k_j is the number of edges of node j in the C partition. For adjacency matrices, $R = C$.

The Erdős-Rényi model generates networks of the same size and with the same number of edges as the observed network, but all edges are randomly distributed among nodes. The second null model distributes edges probabilistically, and the probability of an edge connecting any two nodes depends on the number of edges of both nodes. By using this method, “null model 2” (Bascompte et al. 2003) approximately maintains the original distribution of edges per node.

The Erdős-Rényi model is very general, and differences in modularity between the observed and theoretical networks can result from the unrealistic distribution of edges per node. Conversely, if the degree of modularity of the observed network is greater than that of networks generated using the “null model 2”, this indicates the modular organization of the network results from the connection patterns of each node and not only from the number of edges. Nonetheless, the choice of the appropriate null model will depend on the question of interest. In fact, because MODULAR can utilize large sets of input data, the user can also compute the modularity of the networks generated by other null models by using those as input data. For additional information on the potential uses of null models, please refer to the study published by Gotelli (Gotelli 2000).

References

- Barber, M. J. 2007. Modularity and community detection in bipartite networks. – *Physical Rev. E* 76: 066102.
- Bascompte, J. et al. 2003. The nested assembly of plant-animal mutualistic networks. – *Proc. Natl Acad. Sci. USA* 100: 9383–9387.
- Borgatti, S. P. et al. 2002. Ucinet for windows: Software for social network analysis. Tech. rep., Analytic Technologies, Harvard, MA.
- Carstensen, D. W. et al. 2012. Biogeographical modules and island roles: a comparison of wallacea and the west indies. – *J. Biogeogr.* 39: 739–749.
- Clauset, A. et al. 2004. Finding community structure in very large networks. – *Physical Rev. E* 70: 066111.
- Erdős, P. and Rényi, A. 1959. On random graphs. – *Publicationes Mathematicae Debrecen* 6: 290–297.
- Gotelli, N. J. 2000. Null model analysis of species co-occurrence patterns. – *Ecology* 81: 2606–2621.
- Guimerà, R. and Amaral, L. A. N. 2005. Functional cartography of complex metabolic networks. – *Nature* 433: 895–900.
- Kirkpatrick, S. et al. 1983. Optimization by simulated annealing. – *Science* 220: 671–680.
- Metropolis, N. et al. 1953. Equation of state calculations by fast computing machines. – *Journal of Chemical Physics* 21: 1087–1093.
- Newman, M. E. J. 2006. Finding community structure in networks using the eigenvectors of matrices. – *Physical Rev. E* 74: 036104.
- Newman, M. E. J. and Girvan, M. 2004. Finding and evaluating community structure in networks. – *Physical Rev. E* 69: 026113.
- Wakita, K. and Tsurumi, T. 2007. Finding community structure in mega-scale social networks. In: *Proceedings of the 16th international conference on World Wide Web*. ACM, pp. 1275–1276.