EMCD

A Software Package for Ensemble-based Community Detection in Multilayer Networks

Alessia Amelio, Andrea Tagarelli

DIMES, University of Calabria
Via Pietro Bucci 44, 87036 Rende (CS), Italy

Contact: *tagarelli@dimes.unical.it*

January 28, 2018

1 Overview of the system

EMCD is a software system for finding a community structure in a multilayer network by using an ensemble-based approach. The system has been developed at the Department of Computer Engineering, Modeling, Electronics and Systems of the University of Calabria and is freely distributed. It is implemented in Java version 1.6 and compatible with Unix and Windows operating systems using the Java Virtual Machine.

The algorithm implemented in EMCD is a strategy for combining the solutions found by a community detection algorithm on each layer of the network to compute the final community structure solution of the multilayer network. According to that, the algorithm is based on clustering ensemble, hence it determines a consensus solution from the ensemble solutions found at each layer of the network. The obtained consensus solution brings the information about the membership of the nodes to the communities. Furthermore, it establishes the intra and inter-community connections among the nodes, preserving the topology of the original network. The aim of the algorithm is to find a consensus community structure optimizing the modularity.

EMCD provides three programs corresponding to the three main steps of the algorithm. Input parameters of the programs can be set according to the user's requirements. Furthermore, the source code is available as a collection of methods included in specific classes. Public methods are associated to the main algorithm routines and utility functions which can be externally invoked by the user. Private methods are only used for inner computations inside the public methods, hence they do not have a direct utility for the user. Table

Table 1: Overview of EMCD command-line programs

Operation	JAR program	Method
Partition a network by C-EMCD	generateUpper	consensusCommunities
Partition a network by CC-EMCD	generateLower	consensusCommunitiesLower
Partition a network by M-EMCD	generateGreedy	consensusCommunitiesGreedy

1 shows the three main programs of EMCD.

1.1 Consensus community detection

EMCD software system realizes an implementation of the EMCD framework [1], which is showed in Fig. 1. In this framework, a multilayer network is modeled as a set of unweighted graphs, each representing a specific type of interaction among the nodes corresponding to a given layer. The input of the EMCD framework is a collection of community node memberships, derived from the application of a community detection algorithm on each layer, defining the ensemble $\mathcal{E}$. The EMCD framework is composed of two main blocks: CC-EMCD and M-EMCD.

CC-EMCD applies a clustering ensemble strategy based on the instance-based co-association method for deriving a baseline consensus community structure. As a first step of the procedure, it employs the C-EMCD method for obtaining a community membership solution for the nodes of the graph. It partitions the nodes of the original graph in communities according to the frequency of their connection over the layers. A parameter $\theta \in [0, 1]$ defines the threshold above which the nodes are grouped in the same community. Obtained community membership solution is considered as a topological upper bound of the original graph, because all connections among the nodes are preserved in the graph. Then, CC-EMCD consensus community structure is derived from the community membership solution. It preserves the community membership of the nodes, while reduces the number of connections among the nodes in the graph. Nonetheless, the obtained baseline solution preserves the topology of the original multilayer graph, realizing a topological lower bound of the graph.

Finally, the solution found by CC-EMCD is refined by M-EMCD method for obtaining the consensus community structure. M-EMCD is composed of two steps of intra and inter-community connectivity refinement. The aim is to add intra-community connections from the original graph in the CC-EMCD solution and remove inter-community connections from the CC-EMCD solution if it brings a gain in the modularity. The two steps are iterated until the modularity value becomes stable. In this way, the algorithm finds the best solution by exploring the search space which is limited between the lower and the upper bound solutions.

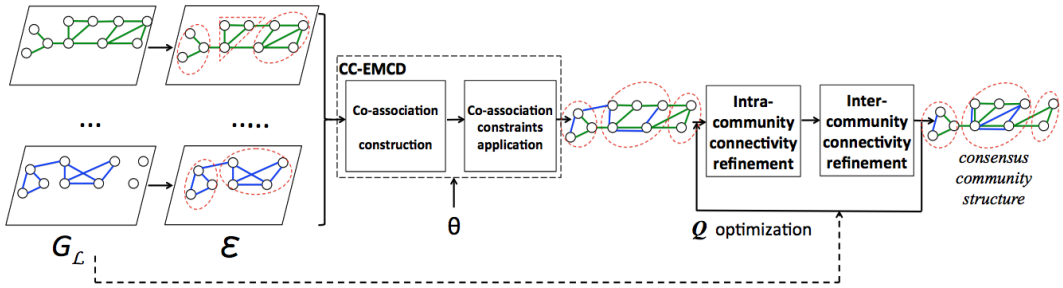

Figure 1: Overview of the EMCD framework [1].

2 Input files

2.1 Multilayer network graph

The multilayer network graph is stored into as many text files as the number of layers. Each file corresponds to the adjacency list representation of the graph of a particular layer; its default name is `adj`i`.txt`, with i=0..#layers-1. Each row in a file corresponds to an entity node and contains its list of neighbors. If a node is not present in layer i, it will have itself as the only neighbor. Nodes are numbered between 1 and #nodes.

2.2 Ensemble of community structures

The ensemble of community structures is stored into as many text files as the number of layers. Each file refers to a structure of non-overlapping communities identified on a particular layer, and contains a column of integer labels, representing the community membership of nodes. The default name of the file is `class`i`.txt`, with i=0..#layers-1. Each row in a file corresponds to an entity node, and contains one community id only. If an entity is not included in a layer, it will be considered as a singleton community, hence it will be assigned a unique label. Community labels may have an arbitrary numbering.

3 Output files

3.1 Consensus community structure

The consensus community structure is stored into a text file, by default named as `consensus.txt`, which is a single column file of integer labels, one for each entity. Each integer represents the community ID which the entity belongs to. Community ids are not necessarily numbered in consecutive order.

3.2 M-EMCD graph

The M-EMCD graph is saved in *ncol* format (<nodeID, nodeID, layerID>), along with: the layer-specific adjacency-list files, the files `layer`i*.txt*, with i=1..#layers, containing each layer of the M-EMCD graph in *ncol* format <nodeID, nodeID>; the files `layer_symmetric`i*.txt*, with i=1..#layers, containing each symmetrized layer of the M-EMCD graph in *ncol* format <nodeID, nodeID>.

3.3 CC-EMCD graph

Output analogous to the above specified for the M-EMCD graph.

4 Programs

4.1 generateUpper

The generateUpper routine runs the method: Consensus.consensusCommunities(int n_layers, int nnodes, double theta, String pathfile, String pathfileout). In particular, it has five input parameters: n_layers which is the number of network layers, nnodes which is the number of nodes of the multigraph, theta which is the threshold value of the C-EMCD method, pathfile which is the path inside the file system containing the network (the network must be represented in terms of layer adjacency lists, $adj_0.txt, ..., adj_{n\_layers-1}.txt$, see Sect 2.1) and the partitioning files for each layer ($class_0.txt, ..., class_{n\_layers-1}.txt$, see Sect 2.2), and pathfileout which is the path inside the file system where the consensus community structure should be saved. The output of the routine is the partitioning file of community membership of the nodes (see Sect. 3) called *consensus.txt* and an integer array of community membership of the nodes (one element at position i represents the community membership of the node $i + 1$).

4.2 generateLower

The generateLower routine runs the method: Consensus.consensusCommunitiesLower(int n_layers, int nnodes, double theta, String pathfile, String pathfileout). In particular, it has five input parameters: n_layers which is the number of network layers, nnodes which is the number of nodes of the multigraph, theta which is the threshold value of the CC-EMCD method, pathfile which is the path inside the file system containing the network (the network must be represented in terms of layer adjacency lists, $adj_0.txt, ..., adj_{n\_layers-1}.txt$, see Sect 2.1) and the partitioning files for each layer ($class_0.txt, ..., class_{n\_layers-1}.txt$, see Sect 2.2), and pathfileout which is the path inside the file system where the consensus community structure should be saved. The output of the routine is specified in Sect. 3.

4.3 generateGreedy

The generateGreedy routine runs the method: Consensus.consensusCommunitiesGreedy (int n_layers, int nnodes, double theta, String pathfile). In particular, it has four input parameters: n_layers which is the number of network layers, nnodes which is the number of nodes of the multigraph, theta which is the threshold value of the M-EMCD method, and pathfile which is the path inside the file system containing the network (the network must be represented in terms of layer adjacency lists, $adj_0.txt, ..., adj_{n\_layers-1}.txt$, see Sect 2.1) and the partitioning files for each layer ($class_0.txt, ..., class_{n\_layers-1}.txt$, see Sect 2.2). The output of the routine is specified in Sect. 3. An example of run is the following:

int n_layers=3;
int nnodes=20;
double theta=0.5;
String pathfile="Desktop/folder1";

Consensus.consensusCommunitiesGreedy(n_layers, nnodes, theta, pathfile);

An example of run in Java is included in the class generateGreedy.java.

5 Data structures

5.1 Graph data structures

- Adjacency structure. It contains the adjacency list of the graph for each layer. It is implemented as an ArrayList L of size equal to the number of layers. Each element L[i] is the adjacency list of layer i, implemented as an HashMap H. Each element H(j) is an HashSet S of neighbors of node j. It is especially used in the C-EMCD method.

- Lower bound graph. It is an HashMap H (key, value), where key is a pair of nodes from the co-association matrix, implemented as an object Pair(node1, node2). The value is the collection of layers involved in the association between node1 and node2 if these nodes are linked by the co-association matrix (implemented as the ArrayList $layer\_int$) or if these nodes are not linked by the co-association matrix (implemented as the ArrayList $layer\_ext$).

- LC_lower. It is used for storing the lower graph in the M-EMCD procedure. It is an HashMap H, where the element H(i) is an HashMap M. Each element M(j) contains an HashSet S of objects Pairs(node1, node2) which are the edges of cluster j at layer i. Edges are only intra-community edges.

- LC_lower_ext. The same of LC_lower, but it contains only inter-community edges.

5.2 Partition data structures

- CC. It is used inside the method for the modularity computation. It is implemented as an HashMap H where an element H(i) is an ArrayList of nodes belonging to the community i.

5.3 Partitioning objective

The method for computing the multilayer modularity is the following: double Consensus.computeModularity(int[]C, int n_layers, String pathfile, double nEdges). It has four input parameters. They are: an integer array C representing the community structure, i.e. the community label of the network node $i + 1$ is contained at position C[i], the integer n_layers which is the number of the network layers, the string pathfile which is the path inside the file system containing the network (the network must be represented in terms of layer adjacency lists, $adj_0.txt, ..., adj_{n\_layers-1}.txt$, see Sect 2.1), and double nEdges which is the number of edges of the symmetric multigraph. The output of the method is a double value representing the modularity.

6 Routines

- Consensus.consensusCommunities(int n_layers, int nnodes, double theta, String pathfile, String pathfileout).
  Input:

  1. n_layers: number of network layers,
  2. nnodes: number of nodes of the multigraph,
  3. theta: threshold value of the C-EMCD method,
  4. pathfile: path inside the file system containing the network and partitioning files for each layer,
  5. pathfileout: path inside the file system where the consensus community structure should be saved.

  Output:

  1. node partitioning file (*consensus.txt*),
  2. integer array of community membership.

- Consensus.consensusCommunitiesLower(int n_layers, int nnodes, double theta, String pathfile, String pathfileout).
  Input:

  1. n_layers: number of network layers,

2. nnodes: number of nodes of the multigraph,
3. theta: threshold value of the C-EMCD method,
4. pathfile: path inside the file system containing the network and partitioning files for each layer,
5. pathfileout: path inside the file system where the consensus community structure should be saved.

Output:

1. node partitioning file (*consensus.txt*),
2. Hash Map H of the lower bound graph.

- Consensus.consensusCommunitiesGreedy (int *n_layers*, int nnodes, double theta, String pathfile).
  Input:

  1. *n_layers*: number of network layers,
  2. nnodes: number of nodes of the multigraph,
  3. theta: threshold value of the C-EMCD method,
  4. pathfile: path inside the file system containing the network and partitioning files for each layer,

  Output:

  1. file of the greedy multigraph in the ncol format,
  2. adjacency file of the greedy graph for each layer,
  3. layer files of the greedy graph in the ncol format,
  4. symmetric layer files of the greedy graph in the ncol format.

- Consensus.update_community(Tuple mod, int i, int key, HashSet<Pair> comm_upper, HashSet<Pair> comm_lower, HashMap<Integer,HashSet<Pair>> Cj_lower) Input:

  1. mod: object of class Tuple containing the modularity value of the lower graph, together with the partial elements for computing the modularity updates,
  2. i: index of the current layer,
  3. key: index of the current community,
  4. comm_upper: current community of the upper graph, represented as a set of node pairs (edges),
  5. comm_lower: corresponding community of the lower graph, represented as a set of node pairs (edges),

6. Cj_lower: set of communities of the current layer in the lower graph, represented as a pair {index of the community, set of node pairs (edges) for that community}.

Output:

1. an object of the class Couples containing the current community index and the modularity update.

- Consensus.update_community_structure(Tuple mod, int layer, int cluster, int[]C, HashSet <Integer>hs, HashMap<Integer, HashMap<Integer,HashSet<Pair>>> lc_lower, HashMap<Integer, HashMap<Integer, HashSet<Pair>>> lc_lower_ext, ArrayList <HashMap<Integer, HashSet<Pair>>> lc_upper_ext) Input:

  1. mod: object of class Tuple containing the modularity value of the lower graph, together with the partial elements for computing the modularity updates,
  2. layer: index of the current layer,
  3. cluster: index of the updated community determining the maximum modularity increase,
  4. C: array of community membership,
  5. hs: set of Integers containing the layer numbers of the lower graph,
  6. lc_lower: for each layer, the HashMap of the pairs {community index, set of node pairs (edges) belonging to that community} in the lower graph,
  7. lc_lower_ext: for each layer, the HashMap of the pairs {community index, set of node pairs (external edges) linking that community with the others} in the lower graph,
  8. lc_upper_ext: for each layer, the HashMap of the pairs {community index, set of node pairs (external edges) linking that community with the others} in the upper graph.

  Output:

  1. an object of the class Couples containing the current community index and the modularity update.

- Consensus.computeModularity(int[]C, int n_layers, String pathfile, double nEdges) Input:

  1. C: array of community membership,
  2. pathfile: path inside the file system where the network is located,
  3. nEdges: number of edges of the symmetric multigraph.

  Output:

  1. double value representing the modularity.

7 Utility functions

- Consensus.printLower(String pathfile, String sep, HashMap<Pair, LowerElement> hm, HashSet<Integer>hs, int nnodes)
  Input:

  1. pathfile: file name in .txt format containing the lower graph,
  2. sep: separator to be used in the file containing the lower graph (e.g. commas),
  3. hm: Hash Map which is the output of consensusCommunitiesLower method (it contains the lower graph),
  4. hs: Hash Set containing the set of layer numbers of the lower graph,
  5. nnodes: node number of the original multigraph.

  Output:

  1. a file .txt containing the lower graph in the ncol format.

- Parser.createNets(String pathfilexc, String pathfileout, String path, String sep)
  Input:

  1. pathfilexc: file name of the multigraph in ncol format,
  2. pathfileout: file name of the renamed and cleaned multigraph,
  3. path: path inside the file system where the layer files should be stored,
  4. sep: separator which is used in the multigraph (e.g. commas).

  Output:

  1. the set of layer files of the renamed and cleaned graph (not symmetric) (e.g. layer0.txt, layer1.txt, etc.)
  2. object of class ContainerFast containing the correspondence between original layers and nodes and renamed layers and nodes.

- Parser.createSymmetric(String pathfile, String pathfilenew)
  Input:

  1. pathfile: file name of the i-th layer of the graph (not symmetric),
  2. pathfilenew: file name of the i-th layer of the graph (symmetric).

  Output:

  1. file containing the symmetric version of the i-th layer of the graph.

- Parser.createAdjList(String pathfile, String pathfilenew, int nodes)
  Input:

  1. pathfile: file name of the i-th layer of the graph (symmetric),
  2. pathfilenew: file name of the adjacency list of the i-th layer of the graph,
  3. nodes: number of network nodes.

  Output:

  1. file of the adjacency list of the i-th layer of the graph.
  2. number of total edges of the graph (if the graph is symmetric, it returns only the effective number of edges).

- Consensus.getLayers(HashMap<Pair,LowerElement>hm)
  Input:

  1. hm: Hash Map containing the elements of the lower graph.

  Output:

  1. Hash Set containing the layer numbers of the lower graph.

8 System requirements & contact info

The ECMD package was developed under a Quad-Core platform, with 2.2 GHz and 16 GB RAM. It is independent from the used operating system.

For any questions about the framework, please contact Andrea Tagarelli (tagarelli@dimes.unical.it).

References

[1] Andrea Tagarelli, Alessia Amelio, Francesco Gullo. Ensemble-based Community Detection in Multilayer Networks. Data Mining and Knowledge Discovery, Springer, Vol. 31, Issue 5, pp. 1506–1543, 2017. DOI: `https://doi.org/10.1007/s10618-017-0528-8`