

```
=====  
== Radatools 5.2 ==  
== from ==  
== radalib-20210720-181118 ==  
==  
== Copyright (c) 2021 by ==  
==  
== Sergio Gomez (sergio.gomez@urv.cat) ==  
== Alberto Fernandez (alberto.fernandez@urv.cat) ==  
==  
== 20/07/2021 ==  
==  
== See LICENSE.txt ==  
=====
```

Contents

- Description
- List of programs
- Communities detection
- Heuristics
- Community detection strategy
- Mesoscales detection
- Network properties
- Comparison of partitions
- Hierarchical clustering
- Usage
- License
- References

Description

Radatools is a set of freely distributed applications to analyze Complex Networks. In particular, it includes programs for community detection and mesoscales search based on the optimization of several modularity-based measures [1,2,3,4].

Radatools [17] is just a set of binary executable programs whose source code is available in Radalib [15]. Radalib is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License version 2.1 as published by the Free Software Foundation.

The program 'Communities_Detection' takes a complex network in Pajek format (*.net) and outputs the best partition found. Since probably you will not have the network in Pajek format, we have added programs to create Pajek network files from text files with lists of links, 'List_To_Net', and from adjacency (or weights) matrices, 'Matrix_To_Net'. Another useful program is 'Connected_Subgraphs', which checks if a network is connected and, if not, creates separate network files for each connected component.

The output of 'Communities_Detection' is a partition in a text format we call 'Lol format' (usually *-lol.txt). It contains information about the number of elements (nodes) in the partition, the number of communities (or lists of elements), the size of each community, and the indices of the elements which belong to each community. The program 'Convert_Lol_To_Clu' converts a partition in Lol format to Pajek (*.clu) format. And 'Reformat_Partitions' replaces the indices of the elements by their respective names in the network Pajek file, allowing also to group elements in a way more appropriate for the analysis of the communities,

or in a more readable form.

If you are working with the standard Newman modularity (weighted or unweighted), you may use the program 'Size_Reduction' to obtain an equivalent network with less nodes (if possible) than the original one, then use 'Communities_Detection', and finally apply 'Size_Reduction_Lol_Expand' to the optimal partition to get a partition of the original (unreduced) network [3].

'Mesoscales_Detection' makes use of the same algorithms implemented in 'Communities_Detection' for the determination of the whole mesoscale of a complex network. Adding a common resistance (in the form of a self-loop) to all nodes, and running its value between 'r_min' and 'r_max', it is possible to find the community structure at all resolution levels, from the macroscale (all nodes in one community) to the microscale (every node in a different community) [6]. The output is a table with the values of the resistance, modularity and number of communities, and the list of different partitions found (*-lols.txt). The program 'Mesoscales_Fine_Tuning' improves the previous results by checking all known partitions for each value of the resistance, and allows the combination of different executions of 'Mesoscales_Detection'. It is also possible to use 'Reformat_Partitions' with the files *-lols.txt.

'Hierarchical_Clustering' performs agglomerative hierarchical clustering of data using both multidendrograms and binary dendrograms. When there are ties in the agglomeration process, the standard approach is to break the ties in an arbitrary way. Multidendrograms ensure the unicity of the dendrogram by just merging as many clusters as necessary when ties are found [12]. This program is able to calculate the multidendrogram and to enumerate (or just count) the different binary dendrograms that can be built from the same data. Additionally, it includes, apart from the standard linkage methods (single, complete, average, centroid, ward, beta-flexible), a parameterized family of space-conserving strategies known as versatile linkage [13]. Visit MultiDendrograms page for an application to calculate and plot multidendrograms with a graphical user interface [16].

Other programs included are: 'Network_Properties', to calculate several of the commonly used properties of a network, e.g. degrees, strengths, clustering coefficients, assortativity, connectedness, shortest path lengths, diameter, betweenness, degree distribution, etc.; 'Compare_Partitions', to obtain measures of the similarity between different partitions, e.g. Jaccard Index, Rand Index, Normalized Mutual Information, Variation of Information, etc.; 'Spanning_Tree', to find the minimum or maximum spanning tree of a graph.

List of programs

The programs available in this version of Radatools and their organization in folders is as follows:

Communities_Detection:

- Communities_Detection
Community detection in complex networks by optimization of modularity, using the following heuristics: (h) exhaustive, (t) tabu, (e) extremal, (s) spectral, (f) fast, (l) louvain, (r) reposition, (b) bootstrapping based on tabu.
- Mesoscales_Detection
Mesoscales detection in complex networks by optimization of modularity for variable common self-loops.
- Mesoscales_Fine_Tuning
Fine Tuning of the mesoscales obtained with Mesoscales_Detection.

Communities_Tools:

- Communities_Network
Given a network and a community, returns the weighted network of communities.
- Compare_Partitions
Calculate similarity and dissimilarity indices between two partitions.
- Convert_Clu_To_Lol
Convert a partition in Pajek format (*.clu) into a partition in our Lol format.
- Convert_Lol_To_Clu
Convert a partition in our Lol format into a partition in Pajek format (*.clu).
- Modularity_Calculation
Calculate the modularity of a partition of a network, detailing the contributions of individual nodes and communities.
- Reformat_Partitions
Reformat partitions in Pajek and Lol formats changing nodes' indices by nodes' names.
- Size_Reduction
Elimination of simple and triangular 'hairs' of a network to speed-up modularity optimization.
- Size_Reduction_Lol_Expand
Convert a partition of a sized reduced network into a partition of the original network.

Data:

- Data_Statistics
Calculate statistics of rows or columns in a data file.
- Data_To_Correlations
Calculate the correlations network of a data file.
- Data_To_Proximities
Calculate many types of proximities (distances or similarities) between rows or columns in a data file.
- Hierarchical_Clustering
Agglomerative hierarchical clustering with multidendrograms and binary dendrograms.

Network_Properties:

- Connected_Subgraphs
Split a network into its (weak or strong) connected components.
- Links_Info
Calculate the degrees and strengths of the nodes attached to each link in a network.
- Network_Properties
Calculate many properties of a network, including connectedness, degrees, strengths, clustering coefficients, assortativities, path lengths, efficiencies, diameters, entropies and betweenness. Handles all kinds of networks, even weighted, directed and signed.

Network_Tools:

- Extract_Subgraphs
Create subgraphs of a graph.
- List_To_Net
Convert a network in list format to Pajek format (*.net).
- Matrix_To_List
Convert a matrix to list format.
- Matrix_To_Net
Convert a network in matrix format to Pajek format (*.net).
- Multiplex_Aggregate
Calculate the aggregate network of a multiplex network.
- Multiplex_Extract_Layers
Extract the layers of a multiplex network.
- Net_To_List
Convert a network in Pajek format (*.net) to list format.
- Net_To_Matrix
Convert a network in Pajek format (*.net) to matrix format.
- Sort_Nodes
Sort nodes of a network randomly or according to degree.
- Spanning_Tree
Calculate the minimum and maximum spanning tree of a graph.
- Symmetrize_Network
Symmetrization of a directed graph.

Communities detection

The program 'Communities_Detection' implements several algorithms for the optimization of modularity [1]. It is prepared to work with

- unweighted [1] and weighted [2] networks
- undirected [1] and directed [3] networks
- positive [1] and signed [4] networks
- bipartite positive [5] and signed networks

The main types of (directed and undirected) modularity optimized are:

- UN: unweighted modularity [1,3]
- WN: weighted modularity [2,3]
- WS: weighted modularity for networks with positive and negative links [4]
- WBPM: weighted bipartite modularity based on path motifs [5]

There is an included document which provides the mathematical description of all the types of modularity available.

Prior to the community detection, it is possible to split the network in its connected components using 'Connected_Subgraphs', and to reduce the size of the network using 'Size_Reduction' [3]. In this last case, only the weighted WN modularity should be used.

The optimization algorithms implemented are:

- (h): exhaustive search
- (t): tabu search [6]
- (e): extremal optimization [7]
- (s): spectral optimization [8]
- (f): fast algorithm [9]
- (l): louvain algorithm [10]

- (r): reposition algorithm
- (b): bootstrapping based on tabu search [6]

There are also several initialization modes:

- (!): start with the best partition found so far
- (:): start with the partition found with the previous heuristics
- (-): start with the default partition of the next heuristics
- (.): start with an 'all isolated' partition
- (+): start with an 'all together' partition

It is possible to combine different optimization algorithms and initialization modes in a single run of the communities detection program. For instance, a possible heuristic is 'trfr', which consists in a tabu search, followed by a reposition fine-tuning, followed by the fast-algorithm, and finally another reposition fine-tuning. It is equivalent to '-t:r:f:r'. Another example could be 's-e!rfr-trfr', where first spectral and extremal are used starting from their defaults (all together), then an 'rfr' is applied to the best of them, and finally a 'trfr' is applied starting from the default of tabu; the output is the best partition found through all the process.

Some heuristics have a stochastic behavior, thus several executions could lead to different optimal partitions. In 'Communities_Detection' you can specify the number of repetitions to execute each of the heuristics, before proceeding to the next one.

If the partition file exists before the execution, it is taken as the initial partition. Thus, running the program twice with heuristics 'trfr' is equivalent to running it once with heuristics 'trfrtrfr'.

There are four levels of verbosity:

- (n): none
- (s): summary
- (p): progress
- (v): verbose

For long runs, 'p' or 'v' should be preferred since they save in temporary files the best intermediate partitions.

The output partitions are given in a list-of-lists text format (files *-lol.txt), which includes information of the number of elements (nodes), number of lists (communities) and, for each community, its size and the list of nodes that belong to it. The tool 'Convert_Lol_To_Clu' converts these partitions to Pajek format (membership of nodes to Communities), and 'Reformat_Partitions' helps to transform partitions to a more readable format.

Heuristics

A short description of each community detection heuristic follows:

- Exhaustive search (h): Only possible for very small networks, about 10 nodes.
- Tabu search (t): Usually the most accurate heuristic for medium sized networks, however it cannot deal with large networks. If you use it, please cite [6].
- Extremal optimization (e): Divisive algorithm. Good tradeoff between accuracy and execution time, and useful for networks too large for tabu search. If you use it, please cite [7].
- Spectral optimization (s): Divisive algorithm. Very fast heuristic and useful for large networks, but with low accuracy. If you use it, please cite [8].
- Fast algorithm (f): Agglomerative algorithm. Fast heuristic but with low accuracy. It may be used as a fine-tuning algorithm when combined with others.

If you use it, please cite [9].

- Reposition algorithm (r): Very fast algorithm, with good accuracy, similar to a Kernighan-Lin optimization.
- Louvain algorithm (l): Very fast algorithm, with good accuracy, similar to reposition. If you use it, please cite [10].
- Bootstrapping algorithm (b): Fine-tuning algorithm based on tabu search. If you use it, please cite [6].

Community detection strategy

The correct selection of heuristics may have an important impact on the quality of the final partition and on the time spent to obtain it. Here come some suggestions, but feel free to experiment with other combinations of heuristics and initialization modes:

- For networks up to 1500 nodes, try 'r-s-e-ll!rfr-trfr' with 1 repetition
- If it succeeds in a reasonable time, erase the previous result and increment the number of repetitions to 3, 5 or 10
- For networks up to 5000 nodes, try 'r-s-e-ll!rfr' with about 2 repetitions
- For networks up to 10000 nodes, try 'r-s-ll!rfr' with 1 repetition
- For larger networks, try 'rfr-llrfr' with 1 repetition

Mesoscales detection

The program 'Mesoscales_Detection' implements the strategy in [6,11] for the determination of the community structure of complex networks at different resolution levels, thus finding the whole mesoscale, from all nodes in one community (macroscale) to every node forming its own community (microscale). Based on the addition of a common self-loop to all nodes, and the optimization of modularity [1,2,3] using the same heuristics explained before, the output of this program is formed by two files:

*table.txt: contains four columns:

- 'r': the resistance (or self-loop)
- 'r - r_min': used in the plots to determine the most stable partitions
- 'Q_r': modularity of the best partition found for the given resistance
- 'N_r': number of modules of the best partition found for the given resistance

*lols.txt: list of different partitions found while scanning the mesoscale (in Lol format), with the value of the resistance at which each partition becomes optimal

In this version of the program you may choose any of the weighted modularity types, but we recommend WS [11] or WN [6], for which the minimum and maximum values of the self-loop are automatically calculated. For unsigned networks and positive self-loops both approaches coincide, but WS solves the problems of WN with negative self-loops in directed networks, and is better adapted to deal with any kind of network. With WN and directed networks it is possible that the partition with all nodes in one community may not appear near 'r_asymp', as explained in [6]. In this case, some of the first partitions found should be discarded, namely those with more modules than the first partition with the minimum number of communities.

The recommended strategy to succeed in the finding of the mesoscale structure is the following:

- First, try 'Communities_Detection' to choose the most adequate optimization heuristic and the number of repetitions, and to estimate the time needed;

- if you want a mesoscale with a granularity of 100 values of the resistance, then 'Mesoscales_Detection' will take about 100 times the execution time of 'Communities_Detection'
- If possible, use Tabu search [6], since it takes advantage of the similarity between partitions at similar resolutions, thus allowing an important speed-up, and a higher quality of the mesoscale found
 - If 'r_min' and 'r_max' differ in several orders of magnitude, use a non-uniform scanning of the resistance, e.g. by setting the parameter 'max_delta_loop_ratio' to 10.0 or higher; in this way you will have a larger resolution for smaller values of 'r - r_min', which is usually the most interesting part of the mesoscale

It is possible to improve the mesoscales found using 'Mesoscales_Fine_Tuning'. This program checks all known partitions at every value of the resistance. The idea is that, during the mesoscales determination, it is common that the changes from one partition to the next one are found at values of resistance slightly higher than optimal, due to the similarity of modularity between both partitions. Moreover, if you run 'Mesoscales_Detection' several times, you can merge all partitions in a single *lols.txt file (or in an additional file with name *lols-extra.txt), and let 'Mesoscales_Fine_Tuning' look for the best possible mesoscales structure.

 Network properties

The program 'Network_Properties' performs the calculation of the following properties:

- Global properties:
 - kind of network (weighted/unweighted, directed/undirected, signed/positive)
 - connectedness (strong, weak, not connected)
 - average and total degree
 - average and total strength
 - minimum and maximum values
 - asymmetry
 - reciprocity
 - average clustering coefficient
 - assortativity
 - average path length
 - diameter
 - efficiency
 - average entropy
- Nodes' properties:
 - degrees
 - strengths
 - self-loop
 - minimum, maximum and average value
 - clustering coefficient
 - average and maximum path lengths
 - efficiency
 - entropy
 - node betweenness
- Edges' properties:
 - edge betweenness
- Degree distribution
- Distances between nodes

In all the cases, the program takes into account the kind of network to decide which properties can be calculated. For instance, strengths and weighted properties only make sense for weighted networks, input and output degrees and strengths are distinguished for directed networks, and total, positive and

negative contributions are separated for signed networks.

Since the calculation of shortest paths (unweighted and weighted) is slow for large networks, you have the option to skip all the properties related with them. It is also possible to skip the calculation of weighted properties if they are not needed.

Comparison of partitions

The program 'Compare_Partitions' calculates the contingency table between two partitions, from which the following similarity and dissimilarity measures are derived:

- Similarity indices:
 - Rand Index
 - Adjusted Rand Index
 - Jaccard Index
 - Fowlkes Mallows Index
 - Normalized Mutual Information Index (arithmetic and geometric)
 - Asymmetric Wallace Index
- Dissimilarity metrics:
 - Mirkin Metric
 - Van Dongen Metric
 - Variation Of Information Metric
 - Normalized Mirkin Metric
 - Normalized Van Dongen Metric
 - Normalized Variation Of Information Metric

Hierarchical clustering

The program 'Hierarchical_Clustering' calculates the agglomerative hierarchical clustering of a proximities (distances or similarities) matrix. It implements the most commonly used algorithms, both unweighted and weighted, and also two parameterized families:

- Single Linkage
- Complete Linkage
- Arithmetic Linkage (UPGMA or Average Linkage)
- Geometric Linkage
- Harmonic Linkage
- Centroid
- Ward
- Versatile Linkage (parameter in range [-1.0, 1.0])
- Beta Flexible (parameter in range [-1.0, 1.0])

Versatile Linkage [13] is a family of space-conserving strategies, which includes several of the most important algorithms as particular cases. The equivalences between clustering algorithms are the following:

- Arithmetic Linkage Unweighted = UPGMA
- Versatile Linkage (param +1.0) = Complete Linkage
- Versatile Linkage (param +0.1) = Arithmetic Linkage
- Versatile Linkage (param 0.0) = Geometric Linkage
- Versatile Linkage (param -0.1) = Harmonic Linkage
- Versatile Linkage (param -1.0) = Single Linkage
- Beta Flexible (param 0.0) = Arithmetic Linkage

It is possible to generate two kinds of dendrograms:

- Multidendrograms [12]

- Binary dendrograms

Multidendrograms solve the non-uniqueness problem of hierarchical clustering found in the standard pair-group algorithms and implementations [12]. This problem arises when two or more minimum distances between different clusters are equal during the amalgamation process. The standard approach consists in choosing a pair, breaking the ties between distances, and proceeds in the same way until the final hierarchical classification is obtained. However, different clusterings are possible depending on the criterion used to break the ties (usually a pair is just chosen at random!), and the user is unaware of this problem.

The Multidendrogram variable-group algorithms group more than two clusters at the same time when ties occur, given rise to a graphical representation called multidendrogram. Their main properties are:

- When there are no ties, the variable-group algorithms give the same results as the pair-group ones
- They always give a uniquely determined solution
- In the multidendrogram representation for the results one can explicitly observe the occurrence of ties during the agglomerative process. Furthermore, the height of any fusion interval (the bands in the program) indicates the degree of heterogeneity inside the corresponding cluster.

If you choose to generate binary dendrograms, there are four options to choose:

- Best: returns the binary dendrogram(s) with maximum cophenetic correlation
- Unsorted: returns all the possible binary dendrograms
- Sorted: generates all the possible binary dendrograms sorted by decreasing value of the cophenetic correlation
- Count: returns only the number of different binary dendrograms

When the number of binary dendrograms is very high [14], the Sorted option may exhaust the available computer memory, so it is convenient to start with Count or Unsorted.

You may use the application MultiDendrograms instead of this tool if you prefer a graphical user interface and plots of the multidendrograms [16]:

- <http://deim.urv.cat/~sergio.gomez/multidendrograms.php>

Usage

Here come some hints on the usage of Radatools:

- Each folder contains program files (*.exe), sample script files (*.bat or *.sh) for each program, and some test files.
- It is highly recommended to have a look at the sample scripts and test files (e.g. with Notepad, vi, nano or emacs) before proceeding with your data.
- The programs may be run directly from command line, but it is more convenient to create script files (*.bat or *.sh) which call the programs with the necessary lists of parameters. Thus, it is recommended that you make a copy of the companion script file (*.bat or *.sh) of the program you want to use, edit it with a text editor, and run it.
- When you run a program without parameters, an 'Usage' message shows the list of expected parameters. The parameters are positional, so be sure you pass the parameters in the right order.
- Read 'LICENSE.txt' to know the conditions to use Radatools.

- Read 'CHANGES.txt' to know the history of updates to Radatools.
- Check periodically the home web site [17] to check if there is a new version of Radatools available.

License

Radatools, Copyright (c) 2021 by
Sergio Gomez (sergio.gomez@urv.cat), Alberto Fernandez (alberto.fernandez@urv.cat)

Radatools is just a set of binary executable programs whose source code is available in Radalib.

Radalib is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License version 2.1 as published by the Free Software Foundation.

Radalib and Radatools are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with Radalib (see LICENSE.txt); if not, see <http://www.gnu.org/licenses/>

References

- [1] M.E.J. Newman and M. Girvan
Finding and evaluating community structure in networks
Physical Review E 69 (2004) 026113
<https://doi.org/10.1103/PhysRevE.69.026113>
- [2] M.E.J. Newman
Analysis of weighted networks
Physical Review E 70 (2004) 056131
<https://doi.org/10.1103/PhysRevE.70.056131>
- [3] Alex Arenas, Jordi Duch, Alberto Fernández and Sergio Gómez
Size reduction of complex networks preserving modularity
New Journal of Physics 9 (2007) 176
<https://doi.org/10.1088/1367-2630/9/6/176>
- [4] Sergio Gómez, Pablo Jensen and Alex Arenas
Analysis of community structure in networks of correlated data
Physical Review E 80 (2009) 016114
<https://doi.org/10.1103/PhysRevE.80.016114>
- [5] Alex Arenas, Alberto Fernández, Santo Fortunato and Sergio Gómez
Motif-based communities in complex networks
Journal of Physics A: Mathematical and Theoretical 41 (2008) 224001
<https://doi.org/10.1088/1751-8113/41/22/224001>
- [6] Alex Arenas, Alberto Fernández and Sergio Gómez
Analysis of the structure of complex networks at different resolution levels
New Journal of Physics 10 (2008) 053039
<https://doi.org/10.1088/1367-2630/10/5/053039>
- [7] Jordi Duch and Alex Arenas
Community detection in complex networks using extremal optimization
Phys. Rev. E 72 (2005) 027104
<https://doi.org/10.1103/PhysRevE.72.027104>

-
- [8] M.E.J. Newman
Modularity and community structure in networks
Proc. Nat. Acad. Sci. USA 103 (2006) 8577
<https://doi.org/10.1073/pnas.0601602103>
- [9] M.E.J. Newman
Fast algorithm for detecting community structure in networks
Physical Review E 69 (2004) 066133
<https://doi.org/10.1103/PhysRevE.69.066133>
- [10] V.D. Blondel, J.-L. Guillaume, R.Lambiotte and E. Lefebvre
Fast unfolding of communities in large networks
Journal of Statistical Mechanics: Theory and Experiment 10 (2008) P10008
<https://doi.org/10.1088/1742-5468/2008/10/P10008>
- [11] Clara Granell, Sergio Gómez and Alex Arenas
Mesoscopic analysis of networks: applications to exploratory analysis and data clustering
Chaos 21 (2011) 016102
<https://doi.org/10.1063/1.3560932>
- [12] Alberto Fernández and Sergio Gómez
Solving non-uniqueness in agglomerative hierarchical clustering using multidendrograms
Journal of Classification 25 (2008) 43-65
<https://doi.org/10.1007/s00357-008-9004-x>
- [13] Alberto Fernández and Sergio Gómez
Versatile linkage: a family of space-conserving strategies for agglomerative hierarchical clustering
Journal of Classification 37 (2020) 584-597
<https://doi.org/10.1007/s00357-019-09339-z>
- [14] Sergio Gómez, Alberto Fernández, Clara Granell and Alex Arenas
Structural patterns in complex systems using multidendrograms
Entropy 15 (2013) 5464-5474
<https://doi.org/10.3390/e15125464>
- [15] Sergio Gómez and Alberto Fernández
Radalib: Ada library and tools for the analysis of Complex Networks and more
<http://deim.urv.cat/~sergio.gomez/radalib.php>
- [16] Sergio Gómez and Alberto Fernández
MultiDendrograms
<http://deim.urv.cat/~sergio.gomez/multidendrograms.php>
- [17] Sergio Gómez and Alberto Fernández
Radatools: Communities detection in complex networks and other tools
<http://deim.urv.cat/~sergio.gomez/radalib.php>