**­Analyzed network parameters**

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Description** | **igraph function** |
| Average degree | In the study of graphs and networks, the **degree** of a node in a network is the number of connections it has to other nodes. This parameter describes the average degree of the given vertices in the network. | np.mean(graph.degree()) |
| Betweenness | **Betweenness** centrality measures the extent to which a vertex lies on paths between other vertices. Vertices with high **betweenness** may have considerable influence within a network by virtue of their control over information passing between others. | betweenness(vertices=None, directed=True, cutoff=None, weights=None, nobigint=True) |
| Average local undirected transitivity | In the unweighted case, the transitivity measures the probability that two neighbors of a vertex are connected. In case of the **average local transitivity**, this probability is calculated for each vertex and then the average is taken. Vertices with less than two neighbors require special treatment, they will either be left out from the calculation or they will be considered as having zero transitivity, depending on the *mode* parameter. The calculation is slightly more involved for weighted graphs; in this case, weights are taken into account according to the formula of Barrat et al. | transitivity\_avglocal\_undirected(mode="nan") |
| Undirected transitivity | Calculates the **global transitivity** (clustering coefficient) of the graph.  The transitivity measures the probability that two neighbors of a vertex are connected. More precisely, this is the ratio of the triangles and connected triplets in the graph. The result is a single real number. Directed graphs are considered as undirected ones. | transitivity\_undirected(mode="nan") |
| Local undirected transitivity | Calculates the **local transitivity** (clustering coefficient) of the given vertices in the graph.  The transitivity measures the probability that two neighbors of a vertex are connected. In case of the local transitivity, this probability is calculated separately for each vertex.  The traditional local transitivity measure applies for unweighted graphs only. When the weights argument is given, this function calculates the weighted local transitivity proposed by Barrat et al (see references). | transitivity\_local\_undirected(vertices=None, mode="nan", weights=None) |
| Density | The **density** of a network is defined as a ratio of the number of edges E to the number of possible edges in a network with N nodes, given (in the case of simple graphs) by the binomial coefficient. | density(loops=False) |
| Diversity | The **structural diversity** index of a vertex is simply the (normalized) Shannon entropy of the weights of the edges incident on the vertex. | diversity(vertices=None, weights=None) |
| Vertex connectivity | The **vertex connectivity** between two given vertices is the number of vertices that have to be removed in order to disconnect the two vertices into two separate components. This is also the number of vertex disjoint directed paths between the vertices (apart from the source and target vertices of course). The vertex connectivity of the graph is the minimal vertex connectivity over all vertex pairs. | vertex\_connectivity(source=-1, target=-1, checks=True, neighbors="error") |
| Jaccard similarity | The **Jaccard similarity** coefficient of two vertices is the number of their common neighbors divided by the number of vertices that are adjacent to at least one of them. | similarity\_jaccard(vertices=None, pairs=None, mode=3, loops=True) |
| Inverse log-weighted similarity coefficient | Each vertex is assigned a weight which is 1 / log(degree). The **log-weighted similarity** of two vertices is the sum of the weights of their common neighbors. | similarity\_inverse\_log\_weighted(vertices=None, mode=3) |
| Dice similarity | **Dice similarity** coefficient of two vertices is twice the number of their common neighbors divided by the sum of their degrees. This coefficient is very similar to the Jaccard coefficient, but usually gives higher similarities than its counterpart. | similarity\_dice(vertices=None, pairs=None, mode=3, loops=True) |
| Assortativity | This coefficient is the correlation between the actual connectivity patterns of the vertices and the pattern expected from the distribution of the vertex types. | assortativity(types1=friend\_graph.vs["sex"], types2=None, directed=True) |
| Assortativity degree | Returns the assortativity of a graph based on vertex degrees. [assortativity\_degree()](http://igraph.org/python/doc/igraph.GraphBase-class.html" \l "assortativity_degree) simply calls [assortativity()](http://igraph.org/python/doc/igraph.GraphBase-class.html" \l "assortativity) with the vertex degrees as types. | assortativity\_degree(directed=True) |
| Nominal assortativity | Assuming that the vertices belong to different categories, this function calculates the assortativity coefficient, which specifies the extent to which the connections stay within categories. The assortativity coefficient is one if all the connections stay within categories and minus one if all the connections join vertices of different categories. For a randomly connected network, it is asymptotically zero. | assortativity\_nominal(types=friend\_graph.vs["sex"], directed=True) |
| Community infomap | Finds the community structure of the network according to the Infomap method of Martin Rosvall and Carl T. Bergstrom.  See [http://www.mapequation.org](http://www.mapequation.org/) for a visualization of the algorithm. | community\_infomap( edge\_weights=None, vertex\_weights=None, trials=10) |
| Community leading eigenvector | Implementation of Newman's eigenvector community structure detection. Each split is done by maximizing the modularity regarding the original network. | community\_leading\_eigenvector(clusters=None, weights=None, arpack\_options=None) |
| Community label propagation | Initially, each vertex is assigned a different label. After that, each vertex chooses the dominant label in its neighbourhood in each iteration. Ties are broken randomly and the order in which the vertices are updated is randomized before every iteration. The algorithm ends when vertices reach a consensus. | community\_label\_propagation(weights=None, initial=None, fixed=None) |
| Community edge betweenness | Community structure detection based on the betweenness of the edges in the network. This algorithm was invented by M Girvan and MEJ Newman, see: M Girvan and MEJ Newman: Community structure in social and biological networks, Proc. Nat. Acad. Sci. USA 99, 7821-7826 (2002).  The idea is that the betweenness of the edges connecting two communities is typically high. So we gradually remove the edge with the highest betweenness from the network and recalculate edge betweenness after every removal, as long as all edges are removed. | community\_edge\_betweenness( clusters=None, directed=True, weights=None) |
| Community spinglass | Finds the community structure of the graph according to the spinglass community detection method of Reichardt & Bornholdt. | community\_spinglass(weights=None, spins=25, parupdate=False, start\_temp=1, stop\_temp=0.01, cool\_fact=0.99, update\_rule="config", gamma=1, implementation="orig", lambda\_=1) |
| Community walk trap | Finds the community structure of the graph according to the random walk method of Latapy & Pons.  The basic idea of the algorithm is that short random walks tend to stay in the same community. The method provides a dendrogram. | community\_walktrap( weights=None, steps=4) |