

MultilayerGraphs.jl: Multilayer Network Science in Julia

Claudio Moroni ^{1,2*} and Pietro Monticone ^{1,2*}

¹ University of Turin, Italy ² Interdisciplinary Physics Team, Italy * These authors contributed equally.

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

MultilayerGraphs.jl is a Julia package for the creation, manipulation and analysis of the structure, dynamics and functions of multilayer graphs.

A multilayer graph consists of multiple subgraphs called *layers* which can be interconnected through [bipartite graphs](#) called *interlayers* composed of the vertex sets of two different layers and the edges between them. The vertices in each layer represent a single set of nodes, although not all nodes have to be represented in every layer.

Formally, a multilayer graph can be defined as a triple $G = (V, E, L)$, where:

- V is the set of vertices;
- E is the set of edges, pairs of nodes (u, v) representing a connection, relationship or interaction between the nodes u and v ;
- L is a set of layers, which are subsets of V and E encoding the nodes and edges within each layer.

Each layer ℓ in L is a tuple (V_ℓ, E_ℓ) , where V_ℓ is a subset of V that represents the vertices within that layer, and E_ℓ is a subset of E that represents the edges within that layer.

MultilayerGraphs.jl is an integral part of the [JuliaGraphs](#) ecosystem extending Graphs.jl ([Fairbanks et al., 2021](#)) so all the methods and metrics exported by Graphs.jl work for multilayer graphs, but due to the special nature of multilayer graphs the package features a peculiar implementation that maps a standard integer-labelled vertex representation to a more user-friendly framework exporting all the objects an experienced practitioner would expect such as nodes (`Node`), vertices (`MultilayerVertex`), layers (`Layer`), interlayers (`Interlayer`), etc.

MultilayerGraphs.jl features multilayer-specific methods and metrics including the global clustering coefficient, the overlay clustering coefficient, the multilayer eigenvector centrality, the multilayer modularity and the Von Neumann entropy.

Finally, MultilayerGraphs.jl has been integrated within the [JuliaDynamics](#) ecosystem so that any `Multilayer(Di)Graph` can be utilised as an argument to the `GraphSpace` constructor in `Agents.jl` ([Datseris et al., 2022](#)).

Statement of Need

Several theoretical frameworks have been proposed to formally subsume all instances of multilayer graphs ([Aleta & Moreno, 2019](#); [Artime et al., 2022](#); [Bianconi, 2018](#); [Boccaletti et al., 2014](#); [Cozzo et al., 2018](#); [M. D. Domenico et al., 2013](#); [M. D. Domenico, 2022](#); [Kivela et al., 2014](#); [Lee et al., 2015](#)).

Multilayer graphs have been adopted to model the structure and dynamics of a wide spectrum of high-dimensional, non-linear, multi-scale, time-dependent complex systems including physi-

cal, chemical, biological, neuronal, socio-technical, epidemiological, ecological and economic networks (Aleta et al., 2022, 2020; Amato et al., 2017; Arruda et al., 2017; Azimi-Tafreshi, 2016; Baggio et al., 2016; Buldú & Porter, 2018; Cozzo et al., 2013; M. D. Domenico, 2017; M. D. Domenico et al., 2016; Estrada & Gómez-Gardeñes, 2014; Gosak et al., 2018; Granell et al., 2013; Lim et al., 2019; Mangioni et al., 2020; Massaro & Bagnoli, 2014; Pilosof et al., 2017; Soriano-Paños et al., 2018; Timóteo et al., 2018).

We have chosen the [Julia language](#) for this software package because it is a modern, open-source, high-level, high-performance dynamic language for technical computing (Bezanson et al., 2017). At the best of our knowledge there are currently no software packages dedicated to the creation, manipulation and analysis of multilayer graphs implemented in the Julia language apart from MultilayerGraphs.jl itself (Moroni & Monticone, 2022).

Main Features

The two main data structures are collections of layers connected through interlayers called MultilayerGraph and MultilayerDiGraph.

The **vertices** of a multilayer graph are representations of one set of distinct objects called Nodes. Each layer may represent all the node set or just a subset of it. The vertices of Multilayer(Di)Graph are implemented via the MultilayerVertex custom type. Each MultilayerVertex encodes information about the node it represents, the layer it belongs to and its metadata.

Both the **intra-layer** and **inter-layer edges** are embedded in the MultilayerEdge struct, whose arguments are the two connected multilayer vertices, the edge weight and its metadata. It's important to highlight that Multilayer(Di)Graphs are weighted and able to store metadata by default (i.e. they have been assigned the IsWeighted and IsMeta traits from [SimpleTraits.jl](#)).

The **layers** are implemented via the Layer struct composed of an underlying graph and a mapping from its integer-labelled vertices to the collection of MultilayerVertices the layer represents. **Interlayers** are similarly implemented via the Interlayer mutable struct, and they are generally constructed by providing the two connected layers, the (multilayer) edge list between them and a graph. This usage of underlying graphs allows for an easier debugging procedure during construction and a more intuitive analysis afterwards allowing the package to leverage all the features of the JuliaGraphs ecosystem so that it can be effectively considered as a real proving ground of its internal consistency.

The Multilayer(Di)Graph structs are weighted and endowed with the functionality to store both vertex-level and edge-level metadata by default so that at any moment the user may add or remove a Layer or specify an Interlayer and since different layers and interlayers could be better represented by graphs that are weighted or unweighted and with or without metadata, it was crucial for us to provide the most general and adaptable structure. A Multilayer(Di)Graph is instantiated by providing the ordered list of layers and the list of interlayers to the constructor. The latter are automatically specified, so there is no need to instantiate all of them.

Alternatively, it is possible to construct a Multilayer(Di)Graph making use of a graph generator-like signature allowing the user to set the degree distribution or the degree sequence and employs graph realisation methods such as the Havel-Hakimi algorithm for undirected graphs (Hakimi, 1962) and the Kleitman-Wang algorithm for directed ones (Kleitman & Wang, 1973).

Multilayer(Di)Graphs structure may be represented via dedicated WeightTensor, MetadataTensor and SupraWeightMatrix structs, all of which support indexing with MultilayerVertices. Once a Multilayer(Di)Graph has been instantiated, its layers and interlayers can be accessed as their properties.

87 For a more comprehensive exploration of the package features and functionalities we strongly
88 recommend consulting the package [documentation](#).

89 Installation and Usage

90 To install MultilayerGraphs.jl it is sufficient to activate the pkg mode by pressing] in the Julia
91 REPL and then run the following command:

```
pkg> add MultilayerGraphs
```

92 In the following code chunks we synthetically illustrate some of the main features outlined in
93 the previous section.

94 Let's begin by importing the necessary dependencies and setting the relevant constants.

```
using Distributions, Graphs, SimpleValueGraphs
using MultilayerGraphs

# Set the number of nodes: objects represented by multilayer vertices
const n_nodes = 100
# Create a list of nodes
const node_list = [Node("node_$(i)") for i in 1:n_nodes]
```

95 Layers and Interlayers

96 We will instantiate layers and interlayers with randomly-selected edges and vertices adopting a
97 variety of techniques. Layers and Interlayers are not immutable, and mostly behave like normal
98 graphs. The reader is invited to consult the [API](#) for more information.

99 Here we define a layer with an underlying simple directed graph using a graph generator-like
100 (or "configuration model"-like) constructor which allows us to specify both the **indegree** and
101 the **outdegree** sequences. Before instantiating each layer we sample the number of its vertices
102 and, optionally, of its edges.

```
n_vertices = rand(1:100) # Number of vertices
layer_simple_directed = layer_simpledigraph( # Layer constructor
    :layer_simple_directed, # Layer name
    sample(node_list, n_vertices; replace=false), # Nodes represented in the layer
    Truncated(Normal(5, 5), 0, 20), # Indegree sequence distribution
    Truncated(Normal(5, 5), 0, 20) # Outdegree sequence distribution
)
```

103 Then we define a layer with an underlying simple weighted directed graph. This is another kind
104 of constructor that allows the user to specify the number of edges to be randomly distributed
105 among vertices.

```
n_vertices = rand(1:n_nodes) # Number of vertices
n_edges = rand(n_vertices:(n_vertices * (n_vertices - 1) - 1)) # Number of edges
layer_simple_directed_weighted = layer_simpleweighteddigraph( # Layer constructor
    :layer_simple_directed_weighted, # Layer name
    sample(node_list, n_vertices; replace=false), # Nodes represented in the layer
    n_edges; # Number of randomly distributed edges
    default_edge_weight=(src, dst) -> rand() # Function assigning weights to edges
)
```

106 Similar constructors, more flexible at the cost of ease of use, enable a finer tuning. The
107 constructor we use below should be necessary only in rare circumstances, e.g. if the equivalent
108 simplified constructor `layer_simplevaldigraph` is not able to infer the correct return types

109 of default_vertex_metadata or default_edge_metadata, or to use and underlying graph
110 structure that isn't currently supported.

```
n_vertices = rand(1:n_nodes) # Number of vertices
n_edges = rand(n_vertices:(n_vertices * (n_vertices - 1) - 1)) # Number of edges
default_vertex_metadata = v -> ("vertex_$(v)_metadata") # Vertex metadata
default_edge_metadata = (s, d) -> (rand(),) # Edge metadata
layer_simple_directed_value = Layer( # Layer constructor
    :layer_simple_directed_value, # Layer name
    sample(node_list, n_vertices; replace=false), # Nodes represented in the layer
    n_edges, # Number of randomly distributed edges
    ValDiGraph(
        SimpleDiGraph{Int64}(),
        vertexval_types=(String,),
        vertexval_init=default_vertex_metadata,
        edgeval_types=(Float64,),
        edgeval_init=default_edge_metadata,
    ),
    Float64;
    default_vertex_metadata=default_vertex_metadata, # Vertex metadata
    default_edge_metadata=default_edge_metadata # Edge metadata
)
```

Create a list of layers

```
layers = [layer_simple_directed, layer_simple_directed_weighted, layer_simple_directed_v
```

111 There are many more constructors the user is encouraged to explore in the package documenta-
112 tion.

113 The interface of interlayers is very similar to that of layers. It is very important to notice that,
114 in order to define a Multilayer(Di)Graph, interlayers don't need to be explicitly constructed
115 by the user since they are automatically identified by the Multilayer(Di)Graph constructor,
116 but for more complex interlayers the manual instantiation is required.

117 Here we define an interlayer with an underlying simple directed graph.

```
n_vertices_1 = nv(layer_simple_directed) # Number of vertices of layer 1
n_vertices_2 = nv(layer_simple_directed_weighted) # Number of vertices of layer 2
n_edges = rand(1:(n_vertices_1 * n_vertices_2 - 1)) # Number of interlayer edges
interlayer_simple_directed = interlayer_simplicedigraph( # Interlayer constructor
    layer_simple_directed, # Layer 1
    layer_simple_directed_weighted, # Layer 2
    n_edges # Number of edges
)
```

The interlayer exports a more flexible constructor too.

```
n_vertices_1 = nv(layer_simple_directed_weighted) # Number of vertices of layer 1
n_vertices_2 = nv(layer_simple_directed_value) # Number of vertices of layer 2
n_edges = rand(1:(n_vertices_1 * n_vertices_2 - 1)) # Number of interlayer edges
interlayer_simple_directed_meta = interlayer_metadigraph( # Interlayer constructor
    layer_simple_directed_weighted, # Layer 1
    layer_simple_directed_value, # Layer 2
    n_edges; # Number of edges
    default_edge_metadata=(src, dst) -> # Edge metadata
        (edge_metadata="metadata_of_edge_from_$(src)_to_$(dst)"),
    transfer_vertex_metadata=true # Boolean deciding layer vertex metadata inheritance
)
```

```
# Create a list of interlayers
interlayers = [interlayer_simple_directed, interlayer_simple_directed_meta]
```

118 Multilayer Graphs

119 In what follows we construct a directed multilayer graph (MultilayerDiGraph).

```
multilayerdigraph = MultilayerDiGraph( # Constructor
    layers,                                # The (ordered) collection of layers
    interlayers;                          # The manually specified interlayers
                                         # The interlayers that are left unspecified
                                         # will be automatically inserted according
                                         # to the keyword argument below
    default_interlayers_structure="multiplex"
    # The automatically specified interlayers will have only diagonal couplings
)

# Layers and interlayer can be accessed as properties using their names
multilayerdigraph.layer_simplevaldigraph
```

120 Then we proceed by showing how to add nodes, vertices and edges to a directed multilayer
 121 graph. The user may add vertices that do or do not represent nodes which are already present
 122 in the multilayergraph. In the latter case, we have to create a node first and then add the
 123 vertex representing such node to the multilayer graph. The vertex-level metadata are effectively
 124 considered only if the graph underlying the relevant layer or interlayer supports them, otherwise
 125 they are discarded. The same holds for edge-level metadata and/or weight.

```
# Create a node
new_node_1 = Node("new_node_1")
# Add the node to the multilayer graph
add_node!(multilayerdigraph, new_node_1)
# Create a vertex representing the node
new_vertex_1 = MV( # Constructor (alias for "MultilayerVertex")
    new_node_1,      # Node represented by the vertex
    :layer_simplevaldigraph, # Layer containing the vertex
    ("new_metadata") # Vertex metadata
)
# Add the vertex
add_vertex!(
    multilayerdigraph, # MultilayerDiGraph the vertex will be added to
    new_vertex_1       # MultilayerVertex to add
)

# Create another node in another layer
new_node_2 = Node("new_node_2")
# Create another vertex representing the new node
new_vertex_2 = MV(new_node_2, :layer_simplesdigraph)
# Add the new vertex
add_vertex!(
    multilayerdigraph,
    new_vertex_2;
    add_node=true # Add the associated node before adding the vertex
)

# Create an edge
new_edge = MultilayerEdge( # Constructor
```

```

new_vertex_1,          # Source vertex
new_vertex_2,          # Destination vertex
("some_edge_metadata") # Edge metadata
)
# Add the edge
add_edge!(
    multilayerdigraph, # MultilayerDiGraph the edge will be added to
    new_edge           # MultilayerVertex to add
)

```

126 Finally we illustrate how to compute a few multilayer metrics such as the global clustering
 127 coefficient, the overlay clustering coefficient, the multilayer eigenvector centrality, and the
 128 multilayer modularity as defined in M. D. Domenico et al. (2013).

```

# Compute the global clustering coefficient
multilayer_global_clustering_coefficient(multilayerdigraph)
# Compute the overlay clustering coefficient
overlay_clustering_coefficient(multilayerdigraph)
# Compute the multilayer eigenvector centrality
eigenvector_centrality(multilayerdigraph)
# Compute the multilayer modularity
modularity(
    multilayerdigraph,
    rand([1, 2, 3, 4], length(nodes(multilayerdigraph)), length(multilayerdigraph.layers)
)

```

129 Related Packages

130 R

131 Here is a list of software packages for the creation, manipulation, analysis and visualisation of
 132 multilayer graphs implemented in the [R language](#):

- 133 ■ [muxViz](#) implements functions to perform multilayer correlation analysis, multilayer central-
 134 ity analysis, multilayer community structure detection, multilayer structural reducibility,
 135 multilayer motifs analysis and utilities to statically and dynamically visualise multilayer
 136 graphs ([D. Domenico et al., 2014](#));
- 137 ■ [multinet](#) implements functions to import, export, create and manipulate multilayer
 138 graphs, several state-of-the-art multiplex graph analysis algorithms for centrality measures,
 139 layer comparison, community detection and visualization ([Magnani et al., 2021](#));
- 140 ■ [mully](#) implements functions to import, export, create, manipulate and merge multilayer
 141 graphs and utilities to visualise multilayer graphs in 2D and 3D ([Hammoud & Kramer,
 142 2018](#));
- 143 ■ [multinets](#) implements functions to import, export, create, manipulate multilayer graphs
 144 and utilities to visualise multilayer graphs ([Lazega et al., 2008](#)).

145 Python

146 Here is a list of software packages for the creation, manipulation, analysis and visualisation of
 147 multilayer graphs implemented in the [Python language](#):

- 148 ■ [MultiNetX](#) implements methods to create undirected networks with weighted or un-
 149 weighted links, to analyse the spectral properties of adjacency or Laplacian matrices and
 150 to visualise multilayer graphs and dynamical processes by coloring the nodes and links
 151 accordingly;

- [PyMNet](#) implements data structures for multilayer graphs and multiplex graphs, methods to import, export, create, manipulate multilayer graphs and for the rule-based generation and lazy-evaluation of coupling edges and utilities to visualise multilayer graphs ([Kivela et al., 2014](#)).

Acknowledgements

This open-source research software project received no financial support.

References

- Aleta, A., Martín-Corral, D., Bakker, M. A., Piontti, A. P. y, Ajelli, M., Litvinova, M., Chinazzi, M., Dean, N. E., Halloran, M. E., Longini, I. M., Pentland, A., Vespignani, A., Moreno, Y., & Moro, E. (2022). Quantifying the importance and location of SARS-CoV-2 transmission events in large metropolitan areas. *Proceedings of the National Academy of Sciences*, 119(26). <https://doi.org/10.1073/pnas.2112182119>
- Aleta, A., Martín-Corral, D., Piontti, A. P. y, Ajelli, M., Litvinova, M., Chinazzi, M., Dean, N. E., Halloran, M. E., Jr, I. M. L., Merler, S., Pentland, A., Vespignani, A., Moro, E., & Moreno, Y. (2020). Modelling the impact of testing, contact tracing and household quarantine on second waves of COVID-19. *Nature Human Behaviour*, 4(9), 964–971. <https://doi.org/10.1038/s41562-020-0931-9>
- Aleta, A., & Moreno, Y. (2019). Multilayer networks in a nutshell. *Annual Review of Condensed Matter Physics*, 10(1), 45–62. <https://doi.org/10.1146/annurev-conmatphys-031218-013259>
- Amato, R., Díaz-Guilera, A., & Kleineberg, K.-K. (2017). Interplay between social influence and competitive strategical games in multiplex networks. *Scientific Reports*, 7(1). <https://doi.org/10.1038/s41598-017-06933-2>
- Arruda, G. F. de, Cozzo, E., Peixoto, T. P., Rodrigues, F. A., & Moreno, Y. (2017). Disease localization in multilayer networks. *Physical Review X*, 7(1). <https://doi.org/10.1103/physrevx.7.011014>
- Artime, O., Benigni, B., Bertagnolli, G., dAndrea, V., Gallotti, R., Ghavasieh, A., Raimondo, S., & Domenico, M. D. (2022). *Multilayer network science*. Cambridge University Press. <https://doi.org/10.1017/9781009085809>
- Azimi-Tafreshi, N. (2016). Cooperative epidemics on multiplex networks. *Physical Review E*, 93(4). <https://doi.org/10.1103/physreve.93.042303>
- Baggio, J. A., BurnSilver, S. B., Arenas, A., Magdanz, J. S., Kofinas, G. P., & Domenico, M. D. (2016). Multiplex social ecological network analysis reveals how social changes affect community robustness more than resource depletion. *Proceedings of the National Academy of Sciences*, 113(48), 13708–13713. <https://doi.org/10.1073/pnas.1604401113>
- Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1), 65–98. <https://doi.org/10.1137/141000671>
- Bianconi, G. (2018). *Multilayer networks*. Oxford University Press. <https://doi.org/10.1093/oso/9780198753919.001.0001>
- Boccaletti, S., Bianconi, G., Criado, R., Genio, C. I. del, Gómez-Gardeñes, J., Romance, M., Sendiña-Nadal, I., Wang, Z., & Zanin, M. (2014). The structure and dynamics of multilayer networks. *Physics Reports*, 544(1), 1–122. <https://doi.org/10.1016/j.physrep.2014.07.001>

- 194 Buldú, J. M., & Porter, M. A. (2018). Frequency-based brain networks: From a multiplex
195 framework to a full multilayer description. *Network Neuroscience*, 2(4), 418–441. https://doi.org/10.1162/netn_a_00033
196
- 197 Cozzo, E., Arruda, G. F. de, Rodrigues, F. A., & Moreno, Y. (2018). *Multiplex networks*.
198 Springer International Publishing. <https://doi.org/10.1007/978-3-319-92255-3>
- 199 Cozzo, E., Baños, R. A., Meloni, S., & Moreno, Y. (2013). Contact-based social contagion in
200 multiplex networks. *Physical Review E*, 88(5). <https://doi.org/10.1103/physreve.88.050801>
- 201 Datseris, G., Vahdati, A. R., & DuBois, T. C. (2022). Agents.jl: A performant and
202 feature-full agent-based modeling software of minimal code complexity. *SIMULATION*,
203 003754972110688. <https://doi.org/10.1177/00375497211068820>
- 204 Domenico, D., Porter, & Arenas. (2014). MuxViz: A tool for multilayer analysis and
205 visualization of networks. *Journal of Complex Networks*, 3(2), 159–176. <https://doi.org/10.1093/comnet/cnu038>
206
- 207 Domenico, M. D. (2017). Multilayer modeling and analysis of human brain networks. *Giga-*
208 *Science*, 6(5). <https://doi.org/10.1093/gigascience/gix004>
- 209 Domenico, M. D. (2022). *Multilayer networks: Analysis and visualization*. Springer International
210 Publishing. <https://doi.org/10.1007/978-3-030-75718-2>
- 211 Domenico, M. D., Granell, C., Porter, M. A., & Arenas, A. (2016). The physics of spreading
212 processes in multilayer networks. *Nature Physics*, 12(10), 901–906. <https://doi.org/10.1038/nphys3865>
213
- 214 Domenico, M. D., Solé-Ribalta, A., Cozzo, E., Kivelä, M., Moreno, Y., Porter, M. A., Gómez,
215 S., & Arenas, A. (2013). Mathematical formulation of multilayer networks. *Physical Review*
216 *X*, 3(4). <https://doi.org/10.1103/physrevx.3.041022>
- 217 Estrada, E., & Gómez-Gardeñes, J. (2014). Communicability reveals a transition to coordinated
218 behavior in multiplex networks. *Physical Review E*, 89(4). <https://doi.org/10.1103/physreve.89.042819>
219
- 220 Fairbanks, J., Besançon, M., Simon, S., Hoffiman, J., Eubank, N., & Karpinski, S. (2021).
221 *JuliaGraphs/graphs.jl: An optimized graphs package for the julia programming language*.
222 <https://github.com/JuliaGraphs/Graphs.jl/>
- 223 Gosak, M., Markovič, R., Dolenšek, J., Rupnik, M. S., Marhl, M., Stožer, A., & Perc, M.
224 (2018). Network science of biological systems at different scales: A review. *Physics of Life*
225 *Reviews*, 24, 118–135. <https://doi.org/10.1016/j.plrev.2017.11.003>
- 226 Granell, C., Gómez, S., & Arenas, A. (2013). Dynamical interplay between awareness and
227 epidemic spreading in multiplex networks. *Physical Review Letters*, 111(12). <https://doi.org/10.1103/physrevlett.111.128701>
228
- 229 Hakimi, S. L. (1962). On realizability of a set of integers as degrees of the vertices of a linear
230 graph. i. *Journal of the Society for Industrial and Applied Mathematics*, 10(3), 496–506.
231 <https://doi.org/10.1137/0110037>
- 232 Hammoud, Z., & Kramer, F. (2018). Mully: An r package to create, modify and visualize
233 multilayered graphs. *Genes*, 9(11), 519. <https://doi.org/10.3390/genes9110519>
- 234 Kivela, M., Arenas, A., Barthélemy, M., Gleeson, J. P., Moreno, Y., & Porter, M. A. (2014).
235 Multilayer networks. *Journal of Complex Networks*, 2(3), 203–271. <https://doi.org/10.1093/comnet/cnu016>
236
- 237 Kleitman, D. J., & Wang, D. L. (1973). Algorithms for constructing graphs and digraphs with
238 given valences and factors. *Discrete Mathematics*, 6(1), 79–88. [https://doi.org/10.1016/0012-365x\(73\)90037-x](https://doi.org/10.1016/0012-365x(73)90037-x)
239

- 240 Lazega, E., Jourda, M.-T., Mounier, L., & Stofer, R. (2008). Catching up with big fish in
241 the big pond? Multi-level network analysis through linked design. *Social Networks*, 30(2),
242 159–176. <https://doi.org/10.1016/j.socnet.2008.02.001>
- 243 Lee, K.-M., Min, B., & Goh, K.-I. (2015). Towards real-world complexity: An introduction to
244 multiplex networks. *The European Physical Journal B*, 88(2). <https://doi.org/10.1140/epjb/e2015-50742-1>
- 246 Lim, S., Radicchi, F., Heuvel, M. P. van den, & Sporns, O. (2019). Discordant attributes of
247 structural and functional brain connectivity in a two-layer multiplex network. *Scientific*
248 *Reports*, 9(1). <https://doi.org/10.1038/s41598-019-39243-w>
- 249 Magnani, M., Rossi, L., & Vega, D. (2021). Analysis of multiplex social networks with r.
250 *Journal of Statistical Software*, 98(8). <https://doi.org/10.18637/jss.v098.i08>
- 251 Mangioni, G., Jurman, G., & Domenico, M. D. (2020). Multilayer flows in molecular networks
252 identify biological modules in the human proteome. *IEEE Transactions on Network Science*
253 *and Engineering*, 7(1), 411–420. <https://doi.org/10.1109/tNSE.2018.2871726>
- 254 Massaro, E., & Bagnoli, F. (2014). Epidemic spreading and risk perception in multiplex
255 networks: A self-organized percolation method. *Physical Review E*, 90(5). <https://doi.org/10.1103/physreve.90.052817>
- 257 Moroni, C., & Monticone, P. (2022). *MultilayerGraphs.jl: A julia package for the creation,*
258 *manipulation and analysis of the structure, dynamics and functions of multilayer graphs.*
259 University of Turin (UniTO); Interdisciplinary Physics Team (InPhyT). <https://doi.org/10.5281/zenodo.7009172>
- 261 Pilosof, S., Porter, M. A., Pascual, M., & Kéfi, S. (2017). The multilayer nature of eco-
262 logical networks. *Nature Ecology & Evolution*, 1(4). <https://doi.org/10.1038/s41559-017-0101>
- 264 Soriano-Paños, D., Lotero, L., Arenas, A., & Gómez-Gardeñes, J. (2018). Spreading processes
265 in multiplex metapopulations containing different mobility networks. *Physical Review X*,
266 8(3). <https://doi.org/10.1103/physrevx.8.031039>
- 267 Timóteo, S., Correia, M., Rodríguez-Echeverría, S., Freitas, H., & Heleno, R. (2018). Multilayer
268 networks reveal the spatial structure of seed-dispersal interactions across the great rift
269 landscapes. *Nature Communications*, 9(1). <https://doi.org/10.1038/s41467-017-02658-y>