




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

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Published: unpublished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

A multilayer graph is, loosely speaking, a collection of “layers” (represented by regular graphs) that also allows for links between the vertices of different layers. The bipartite graphs constituted by the two sets of vertices of two different layers and the edges between them are called “interlayers”. The vertices in each layer represent a single set of nodes, although not all nodes have to be represented in every layer. There are multiple special cases of multilayer graphs, and multiple frameworks have been proposed to explain them all (see Kivela et al. (2014)). Common application of multilayer graphs are social network and epidemiological modeling.

Statement of need

The Julia graph ecosystem, which gravitates around the {Graph.jl}(Fairbanks et al., 2021) package, was lacking an implementation of general multilayer graphs, particularly one that was simultaneously integrated with the main agent-based modeling library, {Agents.jl}(Agents.jl?). Great care has been devoted to seamlessly integrate the package with the existing ecosystem, filling gaps in the latter where necessary: - [Implementation of isdigraphical and fix of isgraphical](#); - [Implementation of Havel-Hakimi and Kleitman-Wang algorithm for simple graph realization](#); - [Better integration of Agents.jl with the graph ecosystem](#); - [Feedback on the state of the graph ecosystem](#).

This resulted in the creation of two API sets: one meant for the end-user, and the other for the developer.

Overview, Internal Design and Package Philosophy

Although being part of the Graphs.jl's ecosystem, due to the special nature of multilayer graphs this package features a peculiar implementation that maps a standard integer-labelled vertex representation to a more user friendly framework that exports all the objects a practitioner would expect (Nodes, MultilayerVertexs, Layers, Interlayers, etc). The details are briefly described hereafter. The package revolves around two data structures, MultilayerGraph and MultilayerDiGraph. As said above, they are collection of layers whose couplings form the edge sets of the so-called interlayers. The vertices of a multilayer graph are representations of one set of distinct objects named Nodes. Each layer may represent all or just part of such set. The vertices of Multilayer(Di)Graph are implemented via the MultilayerVertex custom type. Each MultilayerVertex carries information about the node it represents, the layer it belongs to and its metadata. Edges, both intra- and inter-layer, are embodied in the MultilayerEdge struct, whose fields are the two MultilayerVertexs involved, the edge

weight and its metadata. Note that `Multilayer(Di)Graphs` are weighted and able to carry metadata by default (i.e. they are given the `IsWeighted` and `IsMeta` traits from [SimpleTraits.jl](#)). Layers are implemented via the `Layer` struct, which is constituted by an underlying graph from the `Graphs.jl` ecosystem and a mapping from its integer-labelled vertices to the collection of `MultilayerVertex`s the layer represents. Interlayers are similarly implemented via the `Interlayer` mutable struct, and they are generally constructed by providing the two `Layers`s involved, the (multilayer) edge list between them and an underlying graph. This usage of underlying graphs allows for easy debugging during construction and more intuitive analysis afterwards. It also allows the package to leverage all the features of the ecosystem, and acts as a proving ground of its consistency and coherence. Now we may understand why `Multilayer(Di)Graph` are weighted and able to carry both vertex and edge-level metadata by default: since they are designed so that at any moment the user may add or remove a `Layer` or specify an `Interlayer`, and since it could be that different layers and interlayers are better substantiated by graphs that are weighted or unweighted and with or without metadata, it was necessary to provide a structure capable to adapt to the most general scenario. As specified in the [Future Developments](#) section of the package README, future enhancements may provide more stringent multilayer graphs data structures, by restricting to specific traits, types and/or special cases defined in the literature. A `Multilayer(Di)Graph` is instantiated by providing to the constructor the ordered list of layers and the list of interlayers. The latter are automatically specified, so there is no need to instantiate all of them. Another way of constructing a `Multilayer(Di)Graph` uses a configuration model-like signature: it allows to select the degree distribution or the degree sequence (indegree and outdegree distributions or sequences may be provided separately for the `MultilayerDiGraph`) and uses the Havel-Hakimi algorithm from {Hakimi (1962)}([havelhakimi?](#)) (or {Kleitman and Wang (1973)}[Kleitman1973] for the directed `MultilayerDiGraph`). Please note that, although inspired from BIANCONI??, this is not a complete implementation of a multilayer configuration model: it lacks the capability to specify a different distributions for different groups of layers and/or interlayers (aspects). Once specified, the full API of `Graphs.jl` works on `Multilayer(Di)Graphs` as they were ordinary extensions of the ecosystem. Moreover, multilayer-specific methods or implementations thereof have been developed, mainly drawing from {De Domenico et al, 2013}([DeDomenico?](#)). They include: - Global Clustering Coefficient - Overlay Clustering Coefficient - (Multilayer) Eigenvector Centrality - (Multilayer) Modularity - Von Neumann Entropy

`Multilayer(Di)Graphs` structure may be represented via dedicated `WeightTensor`, `MetadataTensor` and `SupraWeightMatrix` structs, all of which support indexing with `MultilayerVertex`s.

Once a `Multilayer(Di)Graph` has been instantiated, its layers and interlayers may be accessed as they where its properties. In order to simplify the code and improve performance, `Layers` and `Interlayers`s are not fully stored within `Multilayer(Di)Graphs`, only enough information to reconstruct them when accessed as properties is saved, in the form of `LayerDescriptor` and `InterlayerDescriptors`.

Mathematics

Single dollars (\$) are required for inline mathematics e.g. $f(x) = e^{\pi/x}$

Double dollars make self-standing equations:

$$\Theta(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{else} \end{cases}$$

82 You can also use plain \LaTeX for equations

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x) e^{i\omega x} dx \quad (1)$$

83 and refer to [Equation 1](#) from text.

84 Citations

85 Citations to entries in paper.bib should be in [rMarkdown](#) format.

86 If you want to cite a software repository URL (e.g. something on GitHub without a preferred
87 citation) then you can do it with the example BibTeX entry below for ([fidgit?](#)).

88 For a quick reference, the following citation commands can be used: - @author:2001 ->
89 "Author et al. (2001)" - [@author:2001] -> "(Author et al., 2001)" - [@author1:2001;
90 @author2:2001] -> "(Author1 et al., 2001; Author2 et al., 2002)"

91 Figures

92 Figures can be included like this: Caption for example figure. and referenced from text using
93 [section](#) .

94 Figure sizes can be customized by adding an optional second parameter: Caption for example
95 figure.

96 Acknowledgements

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

98 References

- 99 Fairbanks, J., Besançon, M., Simon, S., Hoffiman, J., Eubank, N., & Karpinski, S. (2021).
100 *JuliaGraphs/graphs.jl: An optimized graphs package for the julia programming language*.
101 <https://github.com/JuliaGraphs/Graphs.jl/>
- 102 Kivela, M., Arenas, A., Barthélemy, M., Gleeson, J. P., Moreno, Y., & Porter, M. A. (2014).
103 Multilayer networks. *Journal of Complex Networks*, 2(3), 203–271. [https://doi.org/10.](https://doi.org/10.1093/comnet/cnu016)
104 [1093/comnet/cnu016](https://doi.org/10.1093/comnet/cnu016)