“Using n-layer graph models for representing and transforming knowledge on biological pathways”

DISSERTATION

for the award of the degree

“Doctor rerum naturalium” (Dr. rer. nat.)

Division of Mathematics and Natural Sciences

of the Georg-August-Universität Göttingen

submitted by

Zaynab HAMMOUD

from Kafarmelki, Lebanon

Göttingen 2021

Thesis Committee

Prof. Dr. Frank Kramer

IT-Infrastructure for Translational Medical Research

University of Augsburg

Prof. Dr. Edgar Wingender

Department of Bioinformatics

University Medical Centre Göttingen

Prof. Dr. Stephan Waack

Institute of Computer Science

Georg August University Göttingen

Members of the Examination Board

1st Referee: Prof. Dr. Frank Kramer

2nd Referee: Prof. Dr. Edgar Wingender

Further members of the Examination Board

Prof. Dr. Tim Beißbarth

Department of Medical Bioinformatics

University Medical Centre Göttingen

Prof. Dr. Burkhard Morgenstern

Institute for Microbiology and Genetics, Department of Bioinformatics

Georg August University Göttingen

Prof. Dr. Carsten Damm

Institute of Computer Science

Georg August University Göttingen

Date of oral examination: 04.03.2021

Declaration

“I hereby declare that this dissertation has been written independently and with no other sources and helps than referenced.”

Zaynab Hammoud

Augsburg, 31.01.2021

Acknowledgements

I would like to thank everyone who supported and helped me to make this thesis become a reality.

I would like to express my endless gratitude to the Federal Ministry of Education and Research (BMBF) for giving me this great opportunity by funding this project.

I also gratefully acknowledge the help of my supervisor, Prof. Dr. Frank Kramer, for sharing this whole journey with me, all the way, and step by step. Prof. Kramer has helped me academically and personally. He showed me how to look at life from a new perspective. He also helped me change many aspects of my personality and thus became my idol, and I always look up to him, always being generous, kind, thoughtful, and above all successful, optimistic, always spreading positivity among his students and institute members, which was the greatest help in my work. He kept encouraging me to accomplish more, and believe in myself. I consider myself the luckiest student to have had him as my supervisor and mentor. God bless him and his family.

I would also like to thank my other supervisors, Prof. Dr. Edgar Wingender, and Prof. Dr. Stephan Waack who were of great help, by giving me very perceptive and valuable remarks and comments respectively in the field of Bioinformatics and Mathematics. It is also essential to mention that Prof. Wingender was my ticket to Germany. He granted me the opportunity to prepare my Master's thesis at his department, and he recommended me to Prof. Kramer, by saying "Sie ist sehr rührig und recht smart", which translates to "She is very eager and really smart". I thank you whole-heartedly.

In addition, I thank my lab members, old and new, in Göttingen and Augsburg, for all the help they have given me, especially Florian Auer, who accompanied me in my whole journey and answered all of my questions unconditionally.

Finally, yet importantly, I would like to thank my parents and siblings, for allowing and helping me to go beyond our cultural limits, sending me, by myself, abroad to make my dreams come true, supporting me on my busy days, and encouraging me on my very lonely ones. I thank them for their life-long love and support, and I dedicate my work to them, and my lately passing grandfather.

Many thanks to all. Without your support, this work would not be.

Table of Contents

List of Figures

I. Introduction

I.1. Motivation

Studying human biology has been a subject of interest for many decades. With discoveries and new technologies, new findings and knowledge about the inner-workings and functionalities of cells are acquired. Cell networks are highly complex, due to the many players and molecules interacting continuously to achieve a certain task. To study these networks, it is essential to consider the smallest biological detail level, consisting of DNA, RNA, proteins, and metabolites. More importantly, a great interest has been shown in the manners in which these molecules interact, which form biological networks [1]. For instance, protein-protein interaction (PPI) networks describe functional or physical relationships between two or more proteins [2,3]. In this work, we are interested in cell networks, or pathways, which describe very complex interactions between the different levels of the aforementioned molecules. This complex interplay between molecules is modelled using graphs, which allows the generation of machine-readable formats that can be queried, interpreted, and managed by computer-associated tools [4]. In an abstract formulation of a biological graph, the molecules form the nodes, and their interactions form the edges. A typical example is a PPI network, in which the proteins are represented by the nodes and their multiple interactions by the edges. Such graphical representation can help to systematically predict new information on the topology and functionality of these networks, and thus formulating new hypotheses on new connections and functions. In the omics field, this also provides opportunities to build data integration tools to study these networks at the systems level [4]. Nevertheless, this representation is oversimplified and does not consider the multiple aspects a complex biological network can have. Considering a traditional mathematical graph model does not provide the possibility to explore the various groups of nodes individually or the multiple types of connections between the same vertices. This imposes the need to go beyond and employ a more sophisticated formulation.

The aim of this work is to build a multilayer model defined by an n-layer graph, which can accommodate biological pathways while considering the different biological natures of their constituents. The generated model consists of n layers, to which the molecules participating in a given pathway are assigned. Furthermore, knowledge influencing these pathways is integrated as additional layers to provide a *wholist,* more accurate perspective. The implementation of this model also requires the definition of operations and standard procedures to transform the model. The model with the provided operations is used to generate multiple views by deleting or adding additional layers, nodes, or edges, which helps condense the knowledge in a specific area of interest and remove any erroneous or irrelevant information. Existing tools and parsers are automatically used to obtain and parse the data that will be transformed to generate the multilayer models.

The Multipath project addresses two main challenges faced when studying biological pathways, which are data integration and reproducibility.

I.1.1. Data Integration and Interoperability

The explosion of interest shown in this field resulted in a great number of tools and resources that contain highly interrelated knowledge. Currently, there is a multitude of databases, each containing a single biological aspect, such as protein databases, or types of biological pathways, or even redundant information on the same elements. The main idea is to switch from a *reductionist* into a *wholist* perspective, by inspecting system-level interaction. Since biology, unlike physics and chemistry, is a lawless science, dealing with highly interplaying, constantly evolving actors [1], the inclusion of critically relevant information into a single model is advantageous. It eases the process called, inference, which is transforming experimental results into hypotheses [5].

Typically, multiple databases and analysis tools are needed to perform a single bioinformatics query [6]. Pathway diagrams in particular are considered complex due to many factors, including their size. One of these important factors is the information influencing these pathways that has to be included in the diagrams. Examples of this relevant information are expression and experimental data, literature citations, and drug knowledge [5]. The challenging aspect of this problem also originates from the different incompatible standards and formats used by the knowledge sources [6]. Consequently, implementing tools that allow integration of knowledge and interoperability between these knowledge sources is required.

I.1.2. Reproducibility

In cell-biology, most papers study and focus on individual molecules, and the results and findings formulated by the investigator become hard to reproduce since they are his private property [7]. In bioinformatics and biostatistics, empirical investigation research is based on data and computer algorithms [8]. The normally conducted revision procedure ensures the validity and novelty of the experimental parts of a publication, but rarely the simulations. In fact, the policy of publishing the data alongside the articles was recently adapted by the majority of the journals. From a biologist’s perspective, it is not always realistic to have high bioinformatical skills, especially in the field of visualization. The main problem in most existing tools is the complexity of their implementations, which does not motivate non-experts to spend a significant amount of time learning their usage [5].

In [9], four terms related to reproducibility were defined: reproducibility is the ability to obtain the same results using the same data and code, robustness in which the results are achieved using a different code, replicability using the same code but a different set of data, and generalizability using a completely different code and data. Kim et al. also show that this process does not only depend on the availability of the code and data but also many factors including the language, file format, readability, references, parameter option, and version.

When contributing to the academic field, it is always crucial to publish findings and results that can be replicated and easily used by fellow-researchers. A big challenge faced in the field of pathways is the automation of the construction of pathway diagrams [10], which differ from one publication to the other. In *Reactome*, a single pathway file contains multiple versions of the same pathway, published in different articles. Although these publications present the parts of a pathway, they do not provide any documentation or guidance on the derivation of the presented version from the original. In fact, a study realized by Hothorn et al. [11] showed that 69.6% of the articles published in the 50th volume of the *Biometrical Journal* present simulation studies, of which only 30.4% (17 out of 56 papers) published the data, and 14.3% (8 out of 56 papers) presented the code. In another publication by the same authors [8], a workshop was organized where 34 participants with a degree in statistics or a related field were grouped and asked to reproduce results from two publications, a trivial one with all needed materials and an impossible one with non-published source code. It showed that most groups were able to understand and reproduce some of the findings in the first publication, but none of them were capable to reproduce the second.

In this work, I present a tool that allows the tracking of all modifications applied to a pathway model, with the possibility to invert these steps, in order to re-obtain the original input. Hence, when a certain pathway illustration is used in a publication, the list of applied steps can be published alongside, providing scientists the opportunity to follow-up on the findings of their fellow-scientists.

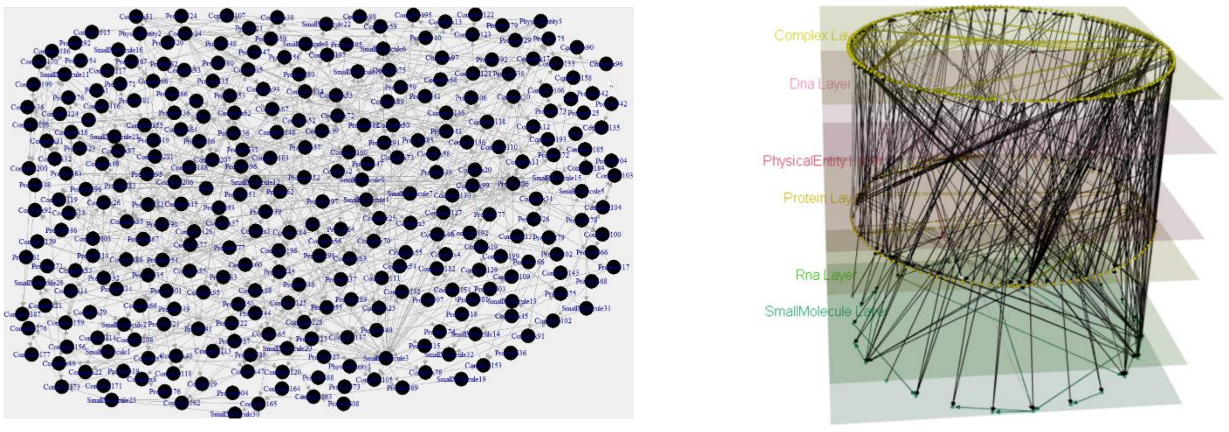
I.2. Graphs

I.2.1. Monolayer Graphs

A graph $G = (V, E)$ is mathematically a non-empty set of nodes V called vertices, connected with a set of relations E called edges, defined by a pair $(v_i, v_j) \in V \times V$ as the end-vertices of the edge e_k [12, 13]. In a normal graph diagram, the vertices are represented with circular nodes or points, and the edges with lines or arrows between the nodes. A graph is called directed when the relations between the adjacent vertices, i.e. incident to a common edge, have a certain direction in which the relationship can be traversed, while in an undirected graph, the edges represent a two-way relationship. The graph can also be weighted, i.e. the edges are assigned weights to indicate the importance or value of certain connections over others. In computer science, graphs are normally represented with adjacency matrices, where the nodes form the columns and rows, and a binary value is assigned to each cell, namely 0 in case the nodes are not adjacent, and 1 in the other case, or the weight of the edge in case of a weighted graph. A subgraph H of a certain graph G consists of a sub-set of vertices and edges, where $V(H) \subseteq V(G)\ and\ E(H) \subseteq E(G)$. G is called the supergraph of H [14].

Graphs are mostly used to describe real-world situations. Directed graphs, for instance, can be used to represent a network of roads, to describe the traffic flow and the orientation of travel. Undirected graphs can be used to represent social relationships in a population of individuals. The relationships can be friendships, neighbourhoods, co-workships, etc., where the direction of edges is immaterial.

This graphical representation shows many inconveniences when the number of nodes and edges becomes relatively huge. When visualizing big networks, a typical problem faced is the hairball effect (Figure 1A), in which the overlapping of nodes and edges becomes considerably high, and results in a highly tangled ball. In this case, the classification of the entities and elements of the graph helps reduce this entanglement, by upgrading the model to support what is called colouring, or layers (Figure 1B). The oversimplified traditional model is thus transformed from mono- to a multilayer model.



(A) (B)

Figure 1 Signaling by Wnt Mono- vs. Multilayer Graphs with Hairball effect. The graph was parsed from Signaling by Wnt BioPAX Level 3, and visualized using the igraph function tkplot (Fig A), and plot3d function in mully (Fig B).

I.2.2. Multilayer Graphs

A multilayer graph is defined by a tuple (V, E, C), where the new set C is introduced as the colours or layers [15]. This concept is employed in many fields and has many interpretations and definitions. Multilayer graphs differ from traditional graphs by the set of layers that groups the nodes, the edges, or both. They provide a better modelling and graphical visualization, by considering the heterogeneity in a given complex network.

I.2.2.1. Abstract Formulation

Multilayer graphs can mainly be classified into two categories, node-coloured and edge-coloured, depending on the chosen definition of layers.

I.2.2.1.1. Node-Coloured Graphs (NCG)

Node-coloured graphs (NCG) are graphs in which the layers represent the various types of nodes. This representation is used to model systems with heterogeneous nodes or entities (Figure 2).

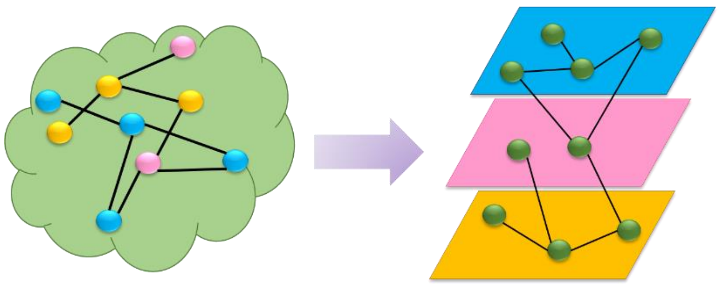


Figure 2 Node-coloured Graphs.

An NCG is defined by a tuple (V, E, C) where V is the set of nodes, E is the set of edges and C is the set of colours or layers.

$$G_N = (V, E, C)\,, \qquad V_\alpha \in V \times C, \qquad E_{\alpha\beta} \in V_\alpha \times V_\beta \;\; where\ \alpha, \beta \in \; C$$

We call an edge connecting two nodes with the same colour intra-edge, and with two different colours inter-edge. The nodes in NCGs are layer-disjoint or in other words mostly assigned one colour.

I.2.2.1.2. Edge-Coloured Graphs (ECG)

In an edge-coloured graph (ECG), the connections between the nodes are heterogeneous. Each aspect of these connections is translated using a colour.

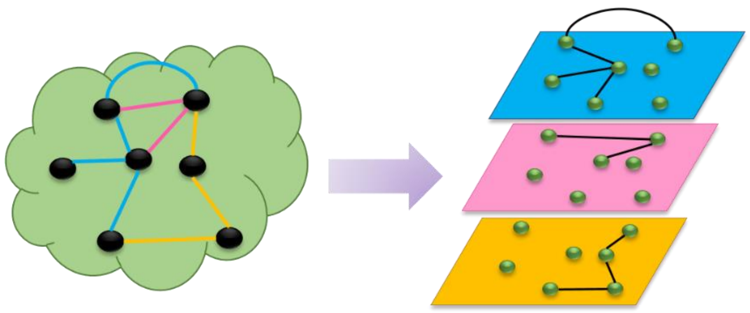


Figure 3 Edge-coloured Graphs.

In most ECGs, the nodes are replicated over the layers (node-aligned), and the edges are distributed over the layers based on their colours (Figure 3).

An ECG is hence defined by a tuple (V, E, C) in the same manner as an NCG. Each layer contains a subset of edges and a subset of or the complete set of nodes.

$$G_E = (V, E, C)$$
$$\{G_\alpha\}_{\alpha=1}^{b} = \{(V_\alpha, E_\alpha)\}_{\alpha=1}^{b}\,, \; \forall \alpha, \beta \in C \quad V_\alpha = V_\beta\,,$$
$$E_\alpha \in V_\alpha \times V_\alpha \times C$$

I.2.2.2. Implementations

The usage of a multilayer networks requires understanding more sophisticated and specific forms and implementations. The abstract model of NCG or ECG can be upgraded to support specific applications. Following are the main existing implementations.

Multiplex and Multilevel networks

These networks are ECG; in a Multiplex graph, each layer contains the complete set of nodes, with subsets of the edges (Figure 4). Multilevel networks are a sub-type of the Multiplex one, with the possibility to assign only certain nodes to the layers (Figure 5).

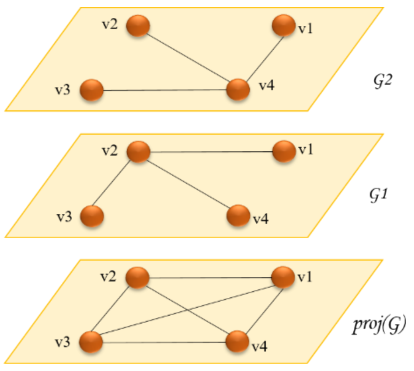


Figure 4 Multiplex Graph.

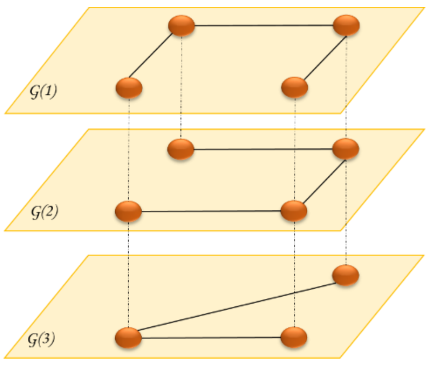


Figure 5 Multilevel Graph.

Interconnected/interacting networks

Interconnected networks consist of multiple networks that are interacting. They are considered NCG since each network contains a different set of nodes, which constitutes the corresponding layer (Figure 6).

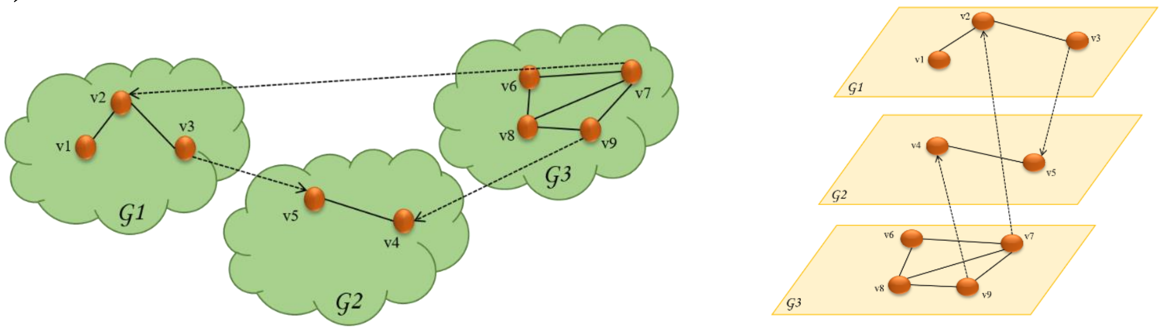


Figure 6 The mono- and multilayer representations of Interconnected networks.

Multi-hypernetworks

Multi-hypernetworks also called hypergraphs, are multiple sets of fully connected nodes. Each set contains several nodes that are connected with a hyper-edge, which is a relation between multiple nodes. The multilayer model is built by creating the various layers that represent the sets and assigning the vertices to their corresponding set. Edges are then added between all the vertices belonging to the same layer (Figure 7).

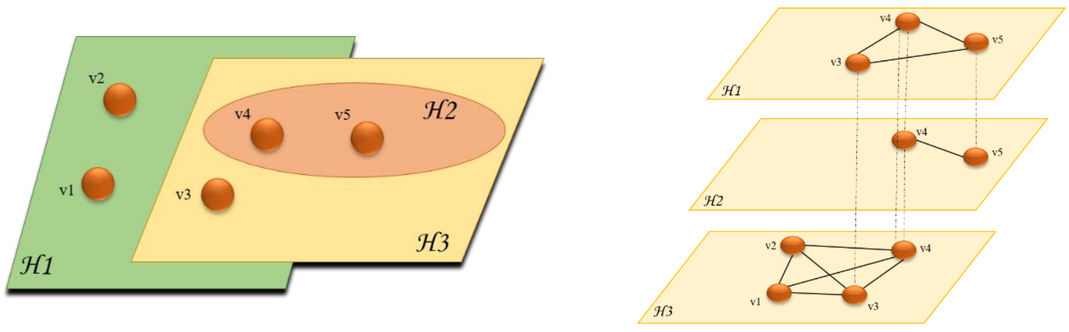


Figure 7 The mono- and multilayer representations of Hypergraphs.

I.2.2.3. Application in Biomedicine

The multilayer framework has been used in recent years to model complex biomedical networks. This usage is justified by the large number of biological elements and relationships in a single biological network, which can be summarized and grouped into layers to allow a better understanding of the structure and dynamics. The layering of biological networks has been employed in different frameworks and for different applications. In my review paper in Appendix A [16], I conducted a literature research on this topic, and I summarized the applications for different types of biological networks. These range from PPI and gene co-expression networks, which are multilayered by respectively grouping the proteins and genes interacting in different settings, such as cells or species, to cell networks, where the layers can represent the multiple molecular types of nodes. Other applications include brain neural networks which can be modelled using Multiplex graphs with different spatial or temporal factors, or frequency bands as layers. The multilayer models have been mostly used in epidemiology and the study of disease spreading, which combines social and biological networks. A typical model is the multiplex SIR model, which consists of 3 layers, namely susceptible, infected and removed.

I.3. Knowledge

I.3.1. Relevant Biological Knowledge

The suffix –omics is normally assigned to a technique used to study, identify and measure a set of biological molecules, such as proteomics for proteomes and genomics for genomes [1]. The interactions between these sets are analysed with network methodologies, used to predict and generate hypotheses. However, the density and complexity of the data constitute the first obstacle in the creation of these networks.

Biological pathways are networks of biological reactions describing how biological molecules interact to accomplish a certain biological function [5]. The main biochemical compounds involved in pathways are proteins, protein complexes, genes, and metabolites. Pathways can be metabolic, developmental, signal-transduction pathways or genetic regulatory circuits. All of these pathways can interact and overlap, by having similar compounds involved in different processes.

There exist over 300 pathway databases that can be grouped into four categories: metabolic pathways, signalling pathways, protein interaction, and gene regulation [17].

Metabolic pathways study the chemical reactions and transport interactions occurring in a cell at different time points, producing energy for vital processes and synthesis of new material [18]. They mainly focus on the chemical transformations of small molecule substrates of enzymes, which are proteins that catalyse chemical reactions [17]. This collected information and how these events correlate is represented by so-called metabolic networks. On the other hand, signalling pathways are regulatory pathways representing one part or sub-process between multiple cells, by propagating series of protein covalent modifications. These databases focus on eukaryotes, which are more complex and diverse [17].

One of the most known pathway databases is the *WikiPathways* open platform, in which the pathways are presented and curated as wiki models that can be modified collaboratively by the entire community. The pathways in *WikiPathways* dispose of separate wiki pages containing the diagram, description references, version history, and components' lists (genes, metabolites, and proteins) [19]. They are encoded in the *XML*-based format *GenMAPP Pathway Markup Language (GPML)*, dedicated to storing graphical information on the pathway diagrams.

PathGuide is a meta-database that displays a pathway resource list with over 700 web-accessible biological pathway and network databases including metabolic and signalling pathways, transcription factor targets, gene regulatory networks, genetic interactions, and PPI and protein-compound interactions in 24 organisms [20]. The data is represented using the *BioPAX, SBML PSI-MI,* and *CellML* standards. The list currently contains 133 signaling pathway databases, and 166 metabolic pathway databases. *Pathway Commons* is one of the metabolic pathways databases that collects biological pathway data from databases through a partnership. The data is represented using the *BioPAX* standard [21]. One of the main partners is *Reactome*.

Kyoto Encyclopedia of Genes and Genomes (KEGG) is a series of 19 databases that contain systems information such as pathway maps in *KEGG PATHWAY*, genomic information such as organisms in *KEGG GENOME*, and chemical information such as drugs in *KEGG DRUG* [22]. The metabolic pathway maps in *KEGG PATHWAY* are represented using the *KEGG XML+SVG (KGML+)* format and can be exported in *BioPAX level 2* format [23]. The map contains different pathway modules, in which the reactions and orthologs are represented by boxes, nodes or molecules are drawn using circles and connected using straight and curved lines.

An example of a signalling network database is *TRANSPATH*®, which is a database system that offers encyclopaedic information and analysis and visualization tools on signal transduction pathways [24]. It also provides cross-references to commonly used databases [25] such as *Ensembl* [26], *RefSeq* [27], *UniProt* [28], *HGNC* [29], and *GO* terms [30]. It is an extension module to the database system *TRANSFAC*® that contains transcription factors, their genomic binding sites, and DNA-binding profiles [31]. The data can be exchanged using the *XML* and *Document Type Definition (DTD)* format. Pathways are considered bipartite graphs, in which the nodes are the molecules and reactions, linked through edges that signify their involvement and participation in one another. They can be visualized using a tool called *PathwayBuilder*®. The latest release of the database is 2020.3, which contains more than 298,000 molecules, 80,000 genes, 98,000 transcription factors, 489,000 reactions, and about 1720 experimentally verified pathways and chains. The number of reactions was also increased by adding 15,202 new binding reactions between human proteins [32].

I.3.2. Knowledge Representation

Creating an abstraction of pathways helps understand their structure and topology and answer questions regarding their functions and organization. Using a pathway standard reduces the time needed to translate the needed knowledge, thus distributing the effort between scientists [17]. In this context, many coding and graphical standards have been implemented to represent the multiple types of knowledge.

Petri nets were extensively used to model biological pathways. In these models, the components of the pathways represent the places, the modification processes represent the transitions or the process nodes, and the interactions are translated using the directional edges or arcs of the *Petri net* [33].

Another graphical representation is the *mEPN* [33], which offers a set of graphical notations to draw pathways and divides the pathway components into four categories:

- *Entity* representing 12 different types of nodes contained in a pathway such as *Protein Complex*, *Protein*, *Gene*, *RNA*, and *Drug*
- *Process* to describe the interactions between the components such as *Binding*, *Cleavage*, *Catalysis*, *Transcription*, etc.
- *Interaction* denoting 6 types of directional edges between the *Entity* and *Process* nodes, such as *Inhibition*, *Action Potential*, and *Catalysis*
- *Cellular Compartment* to indicate the location in which the pathway occurs, such as *Cell Membrane*, *Cytoplasm*, *Nucleus*, and *Mitochondrion*.

The *Systems Biology Graphical Notation (SBGN)* offers a set of graph semantics to be used in the pathway visualization. Three types of diagrams are provided to allow a broader view of the overall system instead of inspecting smaller portions [34].

- *SBGN process diagram (SBGN PD)* describes the changes in the form of biochemical entities during a molecular process. The nodes can be *entity pool*, *process*, *container*, and *reference nodes* connected through *arcs* and *logical operators*. A single node can appear multiple times if it disposes of different states.
- *SBGN entity relationship diagram* is used to describe the influences between the entities. The connections are independent and the entities appear only once. The components of the diagram can be *entity nodes*, *statements*, or *influences*. An example would be a gene regulatory network.
- *SBGN activity flow diagram* is a model with additional rules to eliminate ambiguity and allow a single interpretation of the interactions between the activities instead of the biochemical entities. The idea is to ignore processes and entity states, which are the details that cause the complexity of the processes. This type is usually used for signalling pathways.

Systems Biology Markup Language (SBML) is a standard to represent and exchange general machine-readable models between analysis tools, first developed in 2000 and is being thenceforth improved. It aims to provide a common intermediate format to acknowledge the diversity of computational methods, rather than define a universal language [35]. The *Unified Modeling Language (UML)* is used to define it and create an *eXtensible Markup Language (XML)* representation [34]. The model contains the following types of components: *Compartment*, *Species*, and *Parameter* as attributes to single components or the whole model, *Unit definition,* and *Rule* which are the constraints.

The Proteomics Standards Initiative's Molecular Interaction (PSI-MI) is an *XML*-based format developed by the *Proteomics Standards Initiative (PSI)* to exchange molecular interactions and experimental details. Its goal is to capture and describe information on interactions between biomolecules, including metabolites, proteins, protein complexes, and *DNA*. Other open community standards are also offered by *PSI*, for instance, *MITAB*, *mzidentML*, *mzQuantML*, and *qcML* [36].

Cell Markup Language (CellML) is also an *XML*-based standard that can be used to model metabolic and signal transduction pathways. The model consists of *components* that contain named entities called *variables* and *equations*, *connections* between *variables* in the *components*, *groups*, which represent the named relationships between the *components (encapsulation* and *containment)*, and *units*. The pathways' components are then translated as follows: the reaction is stored as a *component*, which involves a set of *variables* with roles like reactants, product, catalyst, and inhibitor, and finally the *rate* of the equation of the reaction [37].

Another standard in the same field is the *Chemical Markup Language (CML)* for *XML*-based chemical annotations [38].

I.3.2.1. Biological Pathway Exchange (BioPAX)

Biological Pathway Exchange (BioPAX) is an ontology to represent the different aspects of metabolic and signalling pathways and their annotations [10]. It defines a set of terms and descriptions that allow the processing of pathway knowledge by a computer, and the exchange of the pathway data, based on the *Web Ontology Language (OWL)*. This allows the easier exchange of data by storing classes and individuals in a single file in the *XML* format. The *PSI-MI* representation scheme is also followed [10].

BioPAX comprises many pathway abstractions including metabolic pathways, signalling pathways, gene regulatory networks, and genetic and molecular interactions. The first three abstractions are temporally ordered and process-oriented, while the others are static relations. *BioPAX* also includes cross-references to external databases and links to controlled vocabularies, chemical structure, experimental forms of molecules, and sequence feature locations.

Classes, properties, and restrictions are defined to represent the pathways. In the *BioPAX* standard, the individual entities are classified into 6 class types: *Gene*, *Protein*, *DNA*, *RNA*, *Complex* and *Small Molecule*, with the latter 5 being sub-classes of the pool of molecules *PhysicalEntity* (Figure 8).

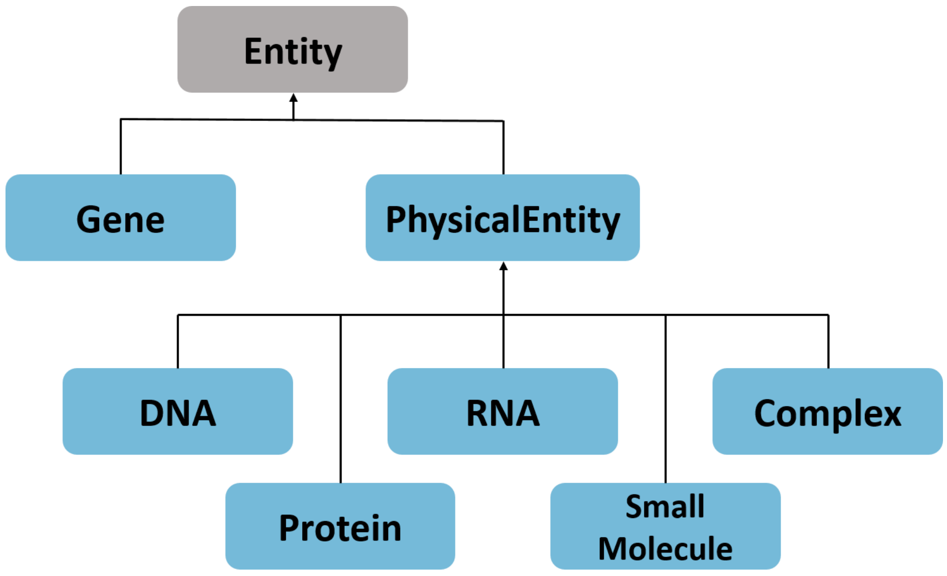


Figure 8 The BioPAX Classes. Adapted from [10].

BioPAX helps translate pathway knowledge into a machine-readable format, which allows an easier usage of data retrieval, analysis, and visualization tools. The language is now supported by many big databases like *Reactome 77*, *KEGG*, and *Pathway Commons*, and important tools such as the visualization tool *Cytoscape* [39] and *PaxTools* [40].

II. Material and Methods

II.1. DrugBank

DrugBank is a comprehensive, freely accessible online database that combines detailed drug data (i.e. chemical, pharmaceutical, and pharmacological) with comprehensive drug targets (i.e. sequence, structure, and pathway) [41].

DrugBank was first published in 2005 [42] as a fully searchable web-enabled, dual-purpose bioinformatics-cheminformatics database, and has been growing since. The first release of *DrugBank* contained more than 4,100 drug entries. An online browser that allowed the browsing of DrugCards, in which the drug entries' information is stored, as well as hyperlinks to external databases were offered. This version was intended to facilitate the discovery of *in-silico* drug targets, prediction of drug interactions, and metabolism.

Later on, other versions were made available and were published. In 2007 *DrugBank* was expanded and *DrugBank* 2.0 was released, containing ~4,900 drug entries and more detailed content [43], followed by three more updates, namely *DrugBank* 3.0 in 2010 [44], *DrugBank* 4.0 in 2013 [45], and *DrugBank* 5.0 being the most recent in 2017 [46]. The latest version of *DrugBank* contains 215 fields per DrugCard compared to 88 in the first release in 2005 (Figure 9).

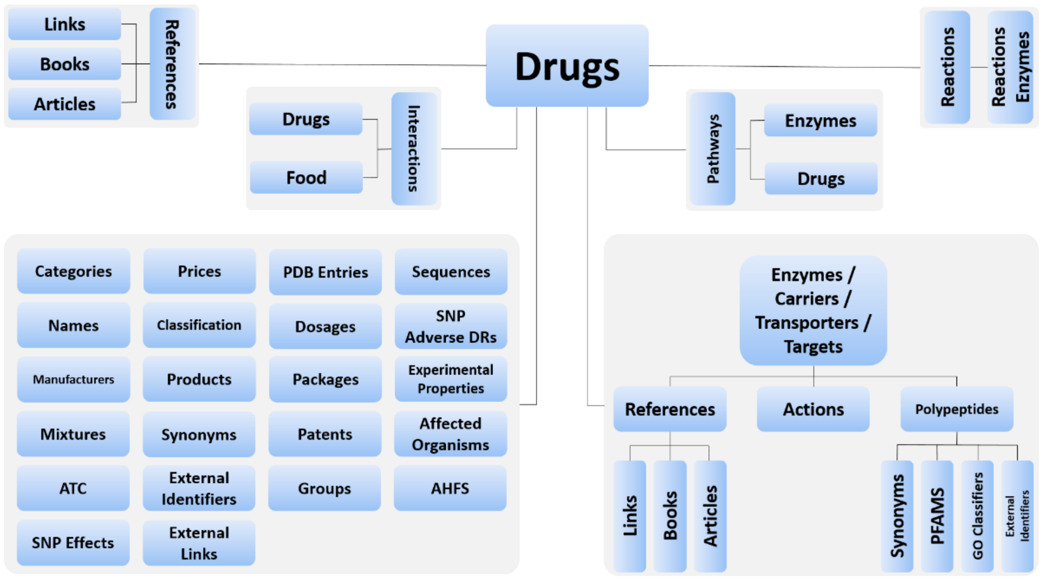


Figure 9 DrugBank Database Structure. The nodes connected to Drugs are the associated information, with their sub-levels.

All drug data entries in *DrugBank* are assigned one of the following groups: approved, vet-approved, experimental, withdrawn, nutraceutical, illicit and investigational. Drug targets are also stored as four different types: protein complexes targets, metabolizing enzymes inhibited/induced or involved in metabolism by drugs, transporters that move drugs into and around the cell, and carriers, which move drugs around the body.

DrugBank is freely accessible through the *DrugBank* Website on drugbank.ca and can be fully and freely downloaded on the aforementioned link.

II.2. Universal Protein Resource (UniProt)

UniProt is a comprehensive resource for protein sequences and annotation data. It combines the effort of the *European Bioinformatics Institute (EMBL-EBI)*, the *Swiss Institute of Bioinformatics (SIB)*, and the *Protein Information Resource (PIR)* as a long-term collaboration to preserve the *UniProt* databases and provide different tasks, like maintenance, database curation and support [28].

UniProt is a collection of three different databases: *UniProt Knowledgebase (UniProtKB)*, *UniProt Reference Clusters (UniRef)*, and *UniProt Archive (UniParc)*.

II.2.1. UniProt Knowledgebase (UniProtKB)

UniProtKB is an expertly and richly curated protein database combining information obtained from two projects: the manually annotated and reviewed project *Swiss-Prot* with over 560,000 reviewed sequences, and the automatically annotated and not reviewed project *TrEMBL* currently containing over 200 million unreviewed sequences. *UniProtKB* contains information on proteomes, currently at ~290,000 proteomes, and protein sets expressed by over 84,000 species [28].

The database is being updated regularly. In their last update in 2019, *UniProt* deprecated a large number of protein sequences based on a redundancy removal process introduced earlier in 2015. The number of entries in all parts of *UniProt*, i.e. *UniProtKB/Swiss-Prot*, *UniRef*, and *UniParc* has increased exponentially, with ~ 120 million entries in *UniProtKB/TrEMBL*, compared to ~ 60 million in 2017 [28].

II.2.2. UniProt Reference Clusters (UniRef)

UniRef is a series of databases, which provide sets of sequence clusters at several resolutions, namely 100%, 90%, and 50%. These clusters aim at organizing *UniProtKB* sequences and reducing redundancy in sequences representing the same space [47]. The sequence sets are generated from *UniProtKB* and *UniParc* consisting of all entries from *UniProtKB*, splice variants from *UniProtKB/Swiss-Prot* entries, and certain entries conforming to certain conditions from *UniParc*.

UniRef100 clusters are generated from sequences at 100% similarity resolution. The longest sequence in a cluster is called seed sequence, and all other sequences belonging to the same *UniRef100* cluster are sub-sequences of the seed sequence. The *UniRef90* clusters are then created from the seed sequences in *UniRef100*, with a similarity of 90%. Similarly, the *UniRef50* clusters are generated from the seed sequences in *UniRef90* at an identity of 50%. The clustering is conducted using the *CD-HIT* algorithm with the different identity thresholds of 100%, 90%, and 50%. This computation was updated in 2015 to include an 80% sequence length overlap threshold with the seed sequence at 90% and 50%. This helped reduce the database size of 2014 against 2007 to 54% compared to 5% for *UniRef100*, 73% compared to 42% for UniRef90, and 88% compared to 70% for *UniRef50* [48].

II.4.3. UniProt Archive (UniParc)

UniParc is the archive of new, revised, and obsolete sequences in *UniProt*, which contains sequences loaded daily from the main publicly available protein sequence databases, like *EMBL/GenBank/DDBJ*, *Ensembl* [26], *RefSeq* [27], PDB, and *MODs* [49]. The archive helps reduce redundancy in storing protein sequences, by assigning a unique, stable *UniParc* identifier to each sequence, which can be used to identify these sequences in external databases. The entries also contain active and retired cross-references to external databases to which the entries are linked.

II.3. Reactome

Reactome is a curated open-source database of metabolic and signalling pathways and reactions in human biology, first released in 2003 [50]. The data is manually annotated and curated by a team of researchers and biologists. It contains cross-references to other resources, such as *UniProt* for proteins [28], *ChEBI* for small molecules [51], *GO* for catalytic activity, cellular compartment and biological processes [30], *PubMed*, and *Ensembl* [26]. *Reactome* also contains putative orthologous reactions in different species, projected electronically from human pathways.

Reactome provides bioinformatics tools. It offers the *Pathway Browser*, which is a tool to view, browse and interact with pathways, as well as analysis tools supporting pathway enrichment and expression analysis [52] such as Analysis Data, Analysis Gene Expression, Species Comparison, and Tissue Distribution.

Reactome uses reactions as a basic unit, in which reactants and products are various and can be proteins, nucleic acids, complexes, small molecules, etc. The reactions contain information on the sub-cellular location in which the reaction takes place, species, and literature evidence supporting them. A group of reactions forms a single pathway [50], which in turn form processes. Each reaction and pathway entry in *Reactome* is specie-specific and contains a tree and a visual map as a part of the *Pathway Browser* displayed at the top of the entry's web page. The page also displays the list of literature references of the reaction or pathway, the participants, and the orthologous events occurring in other species.

Disease annotations were added to the database in the 46th release by adding a disease attribute to the entities and events associated with this disease [53]. The database that was stored in a relational *MySQL* database switched to a non-relational *Neo4j* graph database [54] in *Reactome* version 57 [55]. *Reactome 75* is the latest version of the database, released on December 7th, 2020. It contains a total of 2,477 human pathways, 13,534 reactions, 10,929 proteins, 1,854 small molecules, 414 drugs, and 32,493 literature references [56], compared to 896 human pathways, 5261 reactions, and 5462 proteins in the 10th release in 2005 [50].

The content is available as diagrams and descriptions. The pathway data can be visualized using the *Pathway Browser* provided by *Reactome* analysed and downloaded in different formats, such as *BioPAX* level 2 and 3, *SBML* diagrams, and *PDF*.

II.4. R

R is one of the most popular free, open-source softwares for data analysis and visualization, currently available for *Windows*, *Mac OS X*, and *Linux* operating systems [57]. It is a powerful language and environment to analyse massive data with huge graphics capabilities. It also supports the retrieval of data from multiple data sources, ranging from text files, statistical packages to database management systems. It can be downloaded from the *Comprehensive R Archive Network (CRAN)* at http://cran.r-project.org.

R programming is functional with implicit iterative behaviour that does not encourage coding loops. Hence is the code clearer, and the execution significantly faster and more efficient [58]. Statements are composed of assignments and functions, using the symbol <- to assign values to variables, used bi-directionally, equivalent to the equal sign =. Comments are added using the # symbol [57]. Various data types and structures are included, such as data frames, matrices, vectors, lists, etc. Classes can be defined using *S3* and *S4* classes and methods. For instance, *S3* classes are primitive *R* objects with assigned attributes and class name [59].

An *R* module is called a *package*, which is a collection of related functions, compiled code, and help and data files [57], similar to libraries in *C/C++* and modules in *Perl* [59]. A set of basic-feature packages is included in *R* such as *utils* and *base*, and others are optional to download with more sophisticated graphics, frequently used functions, and statistical modelling methods [59]. These packages are implemented by many contributors and provide different services and functionalities. Most packages are well documented by their developers, which allows an easier and smoother usage. A package is installed to a system or user-defined library, from which it should be loaded before usage. Packages can be downloaded from multiple online repositories, such as *CRAN*, *Bioconductor*, and *GitHub*.

R is updated regularly, with the latest version being *R version 4.0.3* (*Bunny-Wunnies Freak Out*), and *version 4.0.4* (*Lost Library Book*) planned to be released in February 2021.

II.4.1. rBiopaxparser

rBiopaxparser is an *R* package to parse, view and modify level 2 and 3 *BioPAX*-encoded pathway data [60]. It is available on *GitHub* and *Bioconductor*. It uses other data processing and visualization tools such as *RCurl*, *XML*, *Rgraphviz*, *graph*, and *igraph*. The data is parsed using the function *readBiopax()* and stored in a data frame in a *biopax* object (*data.table* format), which is used as an argument passed

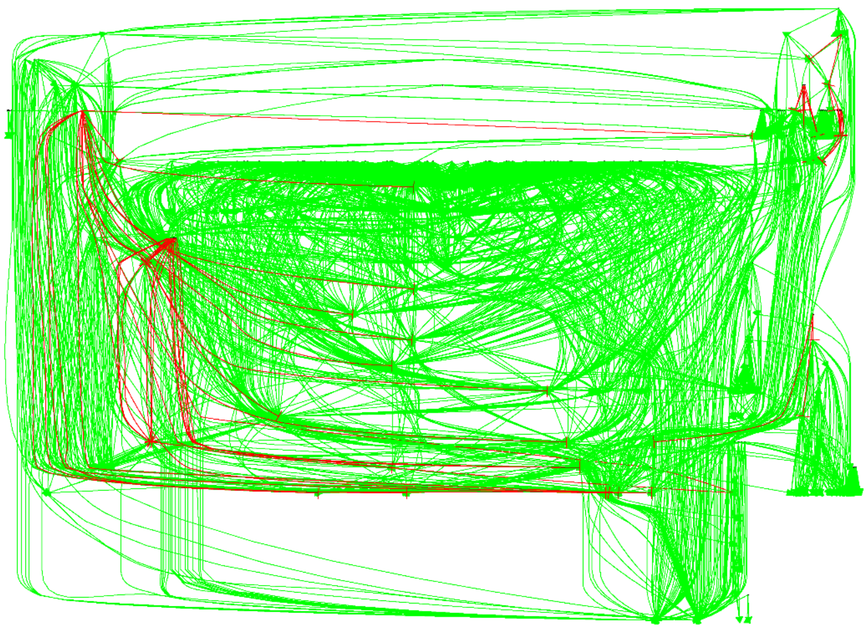


Figure 10 Regulatory Network generated from the Signaling by Wnt BioPAX level 3 file. A red arrow represents an inhibition relation and a green arrow an activation. The nodes are the genes represented with small black points.

to different functions to extract any information concerning the parsed data. This *data.table* object contains components of different classes from the *BioPAX* file, such as pathway, complex, protein, etc. that have different properties and annotations stored conjointly.

Pathways can be transformed to adjacency matrices and regulatory networks (Figure 10) using *pathway2AdjacencyMatrix()* and *pathway2RegulatoryGraph()* respectively. Most importantly, a pathway can be converted to a *graph* object using *pathway2Graph()*, which transforms it into a monolayer network. In Figure 11, a plot of the generated graph object from Signaling by Wnt is shown.

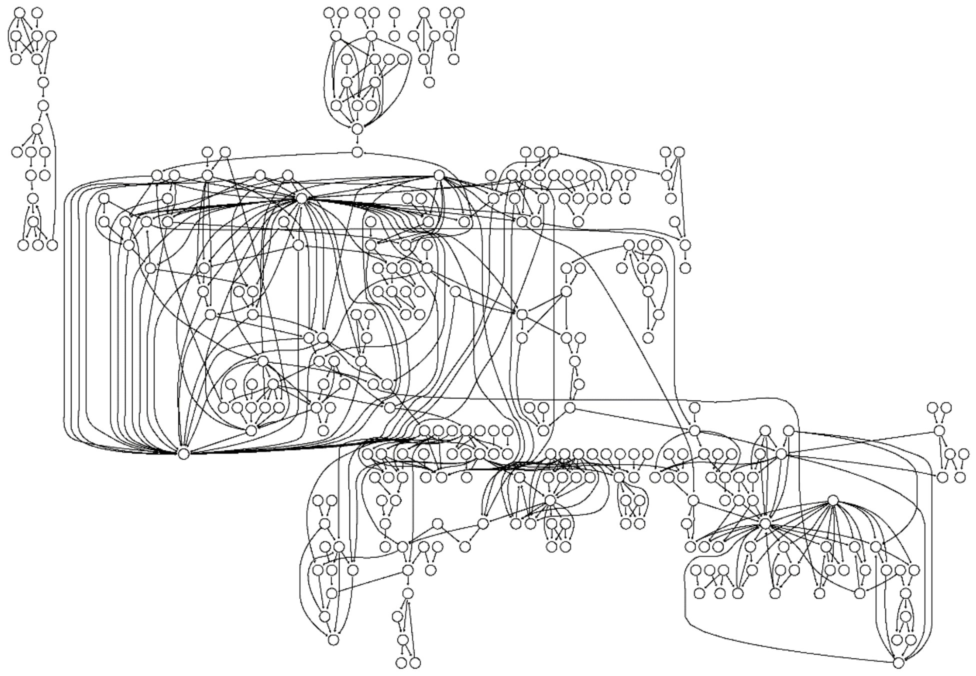


Figure 11 Signaling by Wnt graph model generated from biopax object.

II.4.2. dbparser

dbparser is an *R* package to parse downloaded *XML* data from the *DrugBank* website. The data is parsed using the function *read_drugbank_xml_db()*, which returns a list of data frames, some of which listed below:

- *general_information*: contains the general information about the drug entries, including the primary *DrugBank* key (ex. *DB00001*), the name, state, and type of the drug and date of creation and last modification.
- *pharmacology*: contains pharmacological information about the drugs, including their pharmacodynamics, toxicity, metabolism, and protein binding.
- *interactions*: contains drug-drug interactions, including the name of the drugs and description of the interaction.
- *Small Molecule Pathways (SMP)* information and connections to drugs and proteins in the data frames *pathway*, *pathway_drugs*, and *pathway_enzyme*.
- *groups*: contains information on the drug groups, i.e. approved, experimental, investigational, etc.
- Drug-Protein relations are distributed into four categories, namely carriers, transporters, targets, and enzymes. Each of this category has many corresponding data frames, such as the main one containing the drug-protein relations (*carriers*, *transporters*, *enzymes*, *targets*), actions (e.g. *carriers_actions*) with the corresponding action (binder, inhibitor, agonist, etc.), and information on the polypeptides (*carriers_polypeptides*, *carriers_polypeptides_ex_ident*, *carriers_polypeptides_syn*, *carriers_polypeptides_pfams*, and *carriers_polypeptides_go*).

The package also disposes of functions to parse parts of the downloaded *XML* file, which is relatively big, and its parsing is a slow process. The package is available in *CRAN* at https://cran.r-project.org/web/packages/dbparser/index.html and in the following *GitHub* repository https://github.com/ropensci/dbparser.

II.4.3. UniProt.ws

UniProt.ws is a collection of functions in *R* that allow querying the *UniProt* server to fetch and process the information on *UniProt* entries and cross-referenced data from other databases, namely connections to *DrugBank* drug entries. It provides a *select* interface to fetch information for various species. The default species is *Homo sapiens* with the *NCBI taxId 9606*. This taxonomy can be changed upon the creation of the *UniProt.ws* object using the constructor *UniProt.ws()*, which is the base class to interact with the server and is passed to all other functions. The retrieval of the information is based on combinations of arguments passed to *select()*. *UniProt.ws* offers a list of *columns*, *keytypes,* and *keys*. The first argument specifies the columns of the returned data frame. The keys are the IDs of the databases contained in *UniProt*, with the default being *UniProtKB*, and their types can be retrieved using *keytypes*. The *keytypes* include *DRUGBANK*, *ENSEMBL*, *ENTREZ_GENE*, *HGNC*, *KEGG*, *PHARMGKB* and, *REACTOME*. Most of the keys are also included in the columns, which also contain *GO*, *INTERACTOR*, *KEYWORDS*, etc. For instance, to retrieve the information of the protein entry *P14384* and its *DrugBank* mappings, the following statement is called after initiating the *UniProt.ws* object:

```
select(UniProt.ws_object, keys="P14384",columns=c("UNIPROTKB","DRUGBANK"))
```

This will return a data frame with two columns, the first one with the *UniProtKB* IDs and the second with the *DrugBank* IDs of the drugs related to this protein entry. The package is available on *Bioconductor* at https://www.bioconductor.org/packages/devel/bioc/html/UniProt.ws.html.

III. Cumulative part of the dissertation

III.1. Multilayered Networks: Aspects, Implementation and Application in Biomedicine

This paper was already published in BMC Big Data Analytics in July 2020.

Hammoud, Z*., Kramer, F. Multilayer networks: aspects, implementations, and application in biomedicine. *Big Data Anal* **5**, 2 (2020). doi:10.1186/s41044-020-00046-0

*Corresponding author

III.1.1. Summary and discussion

This is a review paper, in which I summarized my findings of multilayered networks and their usage in Biomedicine. This work was inspired by the challenges that I faced at the beginning of my Ph.D. to learn more about this graphical representation, which goes beyond my knowledge in Graph theory, and monoplex graphs. In the process, I discovered many types and implementations of Multilayered Networks, such as Multiplex, Multilevel, Interdependent Networks, etc. I also faced two challenges: the first one in the terminology of this field, given its young age, and the multiple names the graphs can have, and the second in its application in the different fields of Network Biology. The use of specific implementations in specific applications requires a great amount of research and certainty. Therefore, I presented in this article the state-of-the-art publications that used a multilayer representation to solve a problem in Biomedicine, including epidemiology, PPI networks, neuronal networks, gene expression networks, and cell networks. All information and results that I gathered and obtained are summarized in the introduction in Chapter I.2.2.

III.1.2. Declaration of my contributions

I initiated the literature research about Multilayered Networks with the papers recommended to me by Prof. Dr. Frank Kramer. I conducted this research over 5 months at the beginning of my doctoral thesis to enrich my knowledge in this field, and determine the best design to use for the modelling of pathways in my project. I investigated over 200 papers to assemble this knowledge about multilayer networks. The publications that I read were interdisciplinary and were not all purely biological. Some of these studied social networks, geographical models, and games. These papers also helped me understand how multilayer graphs work and can be used in all fields. The second type of paper is mathematical, in which the models and their formulation were described. I was able to collect many equations that I combined in this review. The last type is bioinformatical and biological articles, which resulted in the last part of the review. This was also a critical phase since it was the base of my model. The projection and comparison of how previous multilayer graph-based research was conducted, led me to the current formulation that I chose, which I consequently implemented in my R package *mully*. I summarized all the results that I obtained and wrote this review paper. Prof. Dr. Kramer has read and proved it for publication.

III.2. mully: an R package to create, modify and visualize multilayered graphs

This paper was already published in MDPI Genes in October 2018.

Hammoud Z*, Kramer F. mully: An R Package to Create, Modify and Visualize Multilayered Graphs. *Genes (Basel). 2018*;9(11):519. Published 2018 Oct 23. doi:10.3390/genes9110519

*Corresponding author

III.2.1. Summary and discussion

In this paper, I present an *R* package that I implemented called *mully*, which stands for **mul**ti**l**a**y**er graphs. The package aims to provide a tool to create graphs with layers, modify them using standard operations, and visualize them. During the literature research phase, I encountered many tools that use a multilayer framework, none of which serves the aim of my thesis. The main problem that I faced was that the tools are case-specific, and cannot be used in other settings. The type of multilayer graphs that I decided to use is the node-coloured abstract definition (Figure 12). I decided to implement a tool that can be used for any use-case, and the structure of the graphs can be modified. The only restriction in *mully* is that the layers are based on the nodes, not the edges. However, the implementation does not specify that the nodes should be layer-disjoint, which means the user can replicate a set of nodes over different layers, which would generate for instance a multiplex network. Thus, I consider this tool a contribution to the *Graph Theory* field, rather than *Bioinformatics* or *Medical Informatics*.

The package extends the *R* package *igraph* [61] and modifies the graph's attributes by adding layers. Upon addition of any node, the layer's name must be specified and the node's name on this layer must be unique. The *mully* object is an *S3 class*, containing the name of the graph, the direction, the layers' information, the list of nodes *V* and edges *E* with their attributes. The package provides the user with a set of standard operations, which are the addition and removal of nodes, edges, and layers (Figure 13).

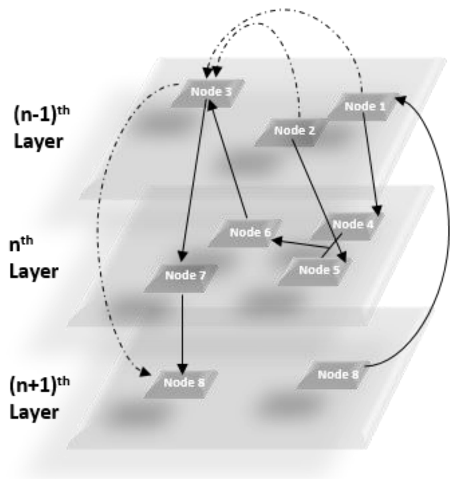


Figure 12 Generic mully Model.

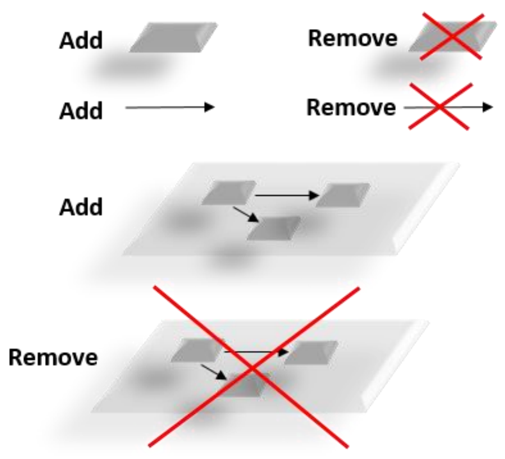


Figure 13 Standard Procedures in mully.

The package also disposes of other features. In the original paper, I presented the track and undo feature as a future idea to be implemented. Later on, the feature was implemented in the package *Multipath*, since I realized it is only an application-specific feature for the *Multipath* project.

Transitivity

The transitivity is one of the most important features *mully* offers. It allows the user to preserve routes and connections between a chain of nodes, by offering the option to add transitive edges upon deletion of certain nodes or layers. Figure 14 shows how the deletion of the node Protein 3 adds a transitive connection between Gene 1 and Metabolite 3 represented by the dotted arrow.

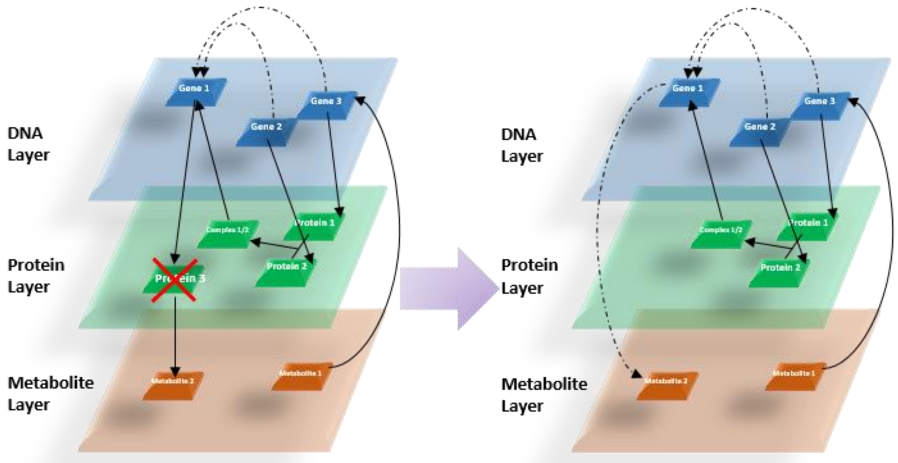


Figure 14 Transitivity Feature in mully.

Visualization

mully objects can be visualized in two ways. The layouts of the graphs are calculated automatically to create the multilayer form. Using the *2D visualizer*, a regular plot is generated, with a random or scaled layout (Figure 15). Each layer is given a fixed coordinate y, and the nodes over the layers are distributed over the x-axis. The function uses the *igraph* plot function and passes random colours and the layout as arguments.

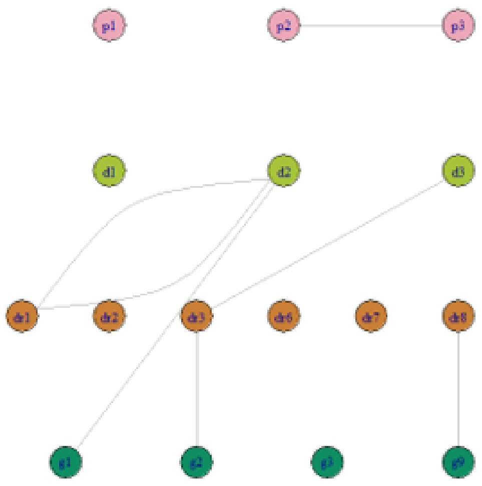


Figure 15 mully 2D Visualizer with Scaled Layout

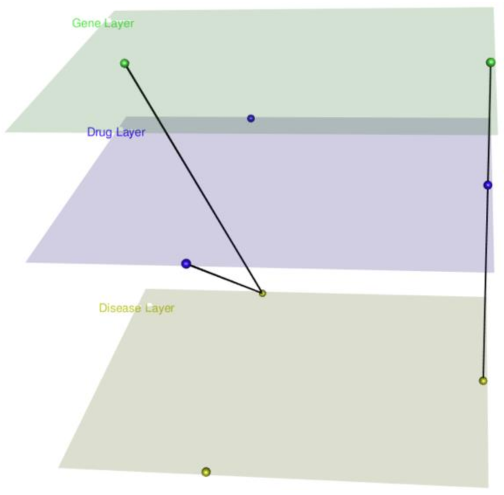


Figure 16 mully 3D Visualizer with a Circular Layout.

The *3D visualizer* is a more advanced one that creates interactive rotatable plots (Figure 16). The *igraph* package provides a primary *rgl* plot function to create these plots. I extended this function by providing a pre-calculated 3D layout based on the positioning of the nodes within the layers. The layout was recently updated to calculate better placements of nodes in big networks. The positioning can be circular or random. The vertices' and edges' sizes and colours can also be specified.

Merge

Another feature in the *mully* package is the possibility to merge two multilayer models based on their layers (Figure 17). As aforementioned, the only restriction in *mully* is the uniqueness of the nodes within single layers. Upon merging two models, nodes belonging to similar layers in both models are combined into one multilayer model.

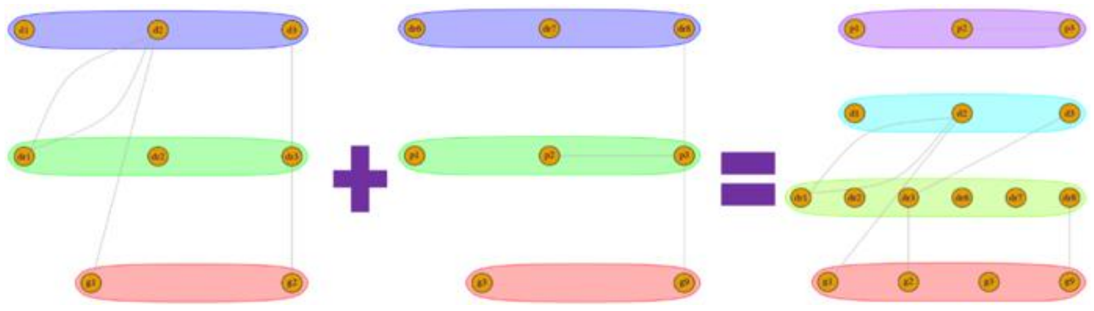


Figure 17 The merge function in mully.

III.2.2. Declaration of my contributions

Prof. Kramer and I designed the model that has to be built. I implemented the *R* Package, with the supervision of Prof. Kramer, who followed the detailed modelling step-by-step. After finishing the implementation, I uploaded *mully* to the *GitHub* repository and wrote this paper. I then submitted it to *MDPI Genes*. The original draft was written by me, and revised and corrected by Prof. Kramer.

Throughout my work, I faced many problems, which required me to update the package and reconsider some of the methods used in the implementation. The most important update since the publication is the *3D visualizer*, to which I added visual planes to represent the layers.

Another important update is the possibility to pre-assign colours to some nodes and edges before the visualization. The origin for this update was a project I was working on, which aimed at annotating *mully* models of breast cancer pathways using *TCGA* gene and protein expression data.

The package is still being maintained and improved until this day, to adapt to new requirements. Lately, in 2020, I added two vignettes (*PDF* and *HTML*) to the package and modified the package to upload it to the *CRAN* repository. The vignette was not published with the paper, but is attached to the *mully* publication in Appendix B. The current version of *mully* is 2.1.31 compared to 1.0 at the date of the publication.

III.3. Multipath: an R package to generate integrated reproducible models

This paper was already published in MDPI Biology in December 2020.

Hammoud Z*, Kramer F. Multipath: An R Package to Generate Integrated Reproducible Pathway Models. *Biology*. 2020 Dec;9(12):483. doi:10.3390/biology9120483

*Corresponding author

III.3.1. Summary and discussion

This was the third part of my Ph.D., in which I implemented an *R* package I called *Multipath*. The aim of *Multipath* is to generate reproducible multilayer pathway models.

Multipath has two major features; the first feature is the transformation *BioPAX*-encoded pathways into multilayer graphs, where other influencing knowledge can be added. The package uses multiple other packages previously implemented (Figure 18).

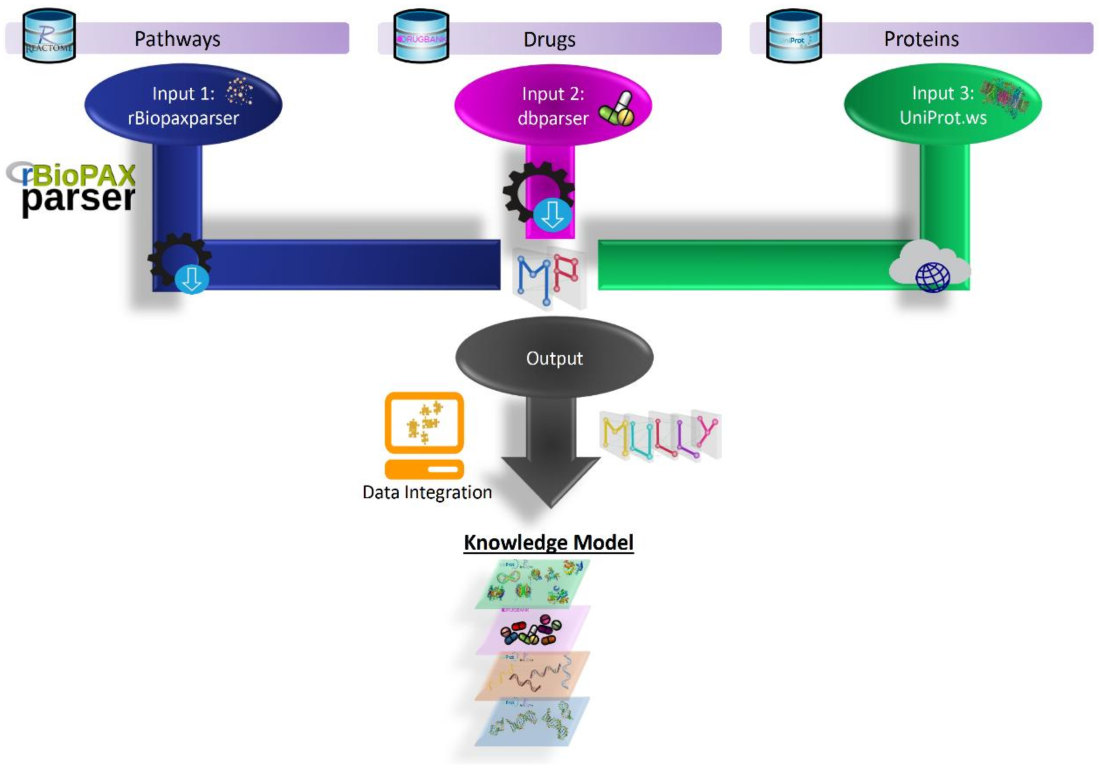


Figure 18 Package dependencies in Multipath.

For parsing the data, I used the *R* package implemented by Prof. Kramer called *rBiopaxparser*, which transforms the parsed data into monolayer graph objects using the package *graphNEL*. I implemented a function to extract the classes of the nodes from these graphs and use them as colours or layers. The resulting multilayer graph is node-coloured and the nodes are layer-disjoint. The multilayer models generated by *Multipath* contain hence up to 6 layers, with the *PhysicalEntity* preserved as a class type, and the *Gene* and *DNA* as a single layer.

On the other hand, the second feature of *Multipath* is the documentation of all modifications applied to a certain multilayer pathway model and the generation of different views. After transforming a *BioPAX*-encoded pathway into a multilayer model, a *pathwayView* object can be created, in which the modifications applied to the pathway are stored. The *pathwayView* stores the list of steps, as well as the original and modified version of the graph. These steps can be reversed to re-obtain previous versions of the same graph. The aim of this feature is to realize the possibility to reproduce specific forms of a given pathway. This workflow is summarized in Figure 19.

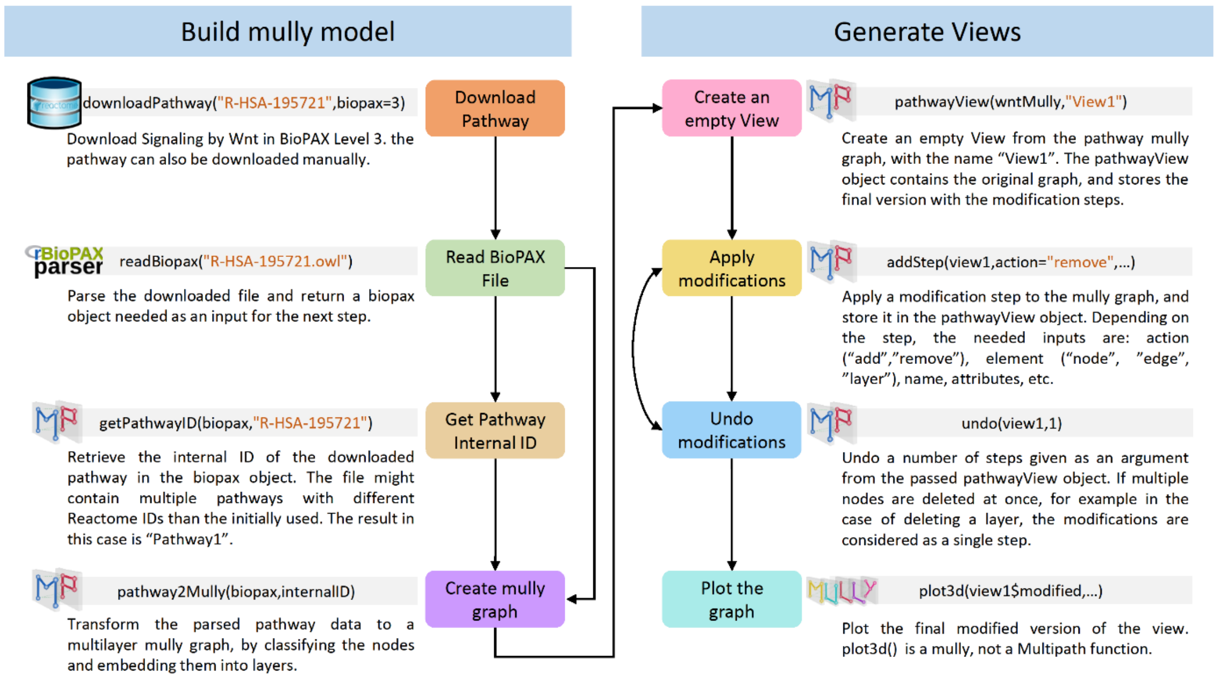

Build mully model
Generate Views
downloadPathway("R-HSA-195721",biopax=3)
Download Signaling by Wnt in BioPAX Level 3. the pathway can also be downloaded manually.
Download Pathway
readBiopax("R-HSA-195721.owl")
Parse the downloaded file and return a biopax object needed as an input for the next step.
Read BioPAX File
getPathwayID(biopax,"R-HSA-195721")
Retrieve the internal ID of the downloaded pathway in the biopax object. The file might contain multiple pathways with different Reactome IDs than the initially used. The result in this case is "Pathway1".
Get Pathway Internal ID
pathway2Mully(biopax,internalID)
Transform the parsed pathway data to a multilayer mully graph, by classifying the nodes and embedding them into layers.
Create mully graph
Create an empty View
pathwayView(wntMully,"View1")
Create an empty View from the pathway mully graph, with the name "View1". The pathwayView object contains the original graph, and stores the final version with the modification steps.
Apply modifications
addStep(view1,action="remove",...)
Apply a modification step to the mully graph, and store it in the pathwayView object. Depending on the step, the needed inputs are: action ("add","remove"), element ("node", "edge", "layer"), name, attributes, etc.
Undo modifications
undo(view1,1)
Undo a number of steps given as an argument from the passed pathwayView object. If multiple nodes are deleted at once, for example in the case of deleting a layer, the modifications are considered as a single step.
Plot the graph
plot3d(view1$modified,...)
Plot the final modified version of the view. plot3d() is a mully, not a Multipath function.

Figure 19 The workflow to generate multilayer reproducible graphs from BioPAX-encoded pathways.

Further information on specific nodes can be added using other *Multipath* functions. *Multipath* currently offers functions to fetch information on proteins from *UniProt* using the *R* web service package called *UniProt.ws*. The diagram in Figure 20 shows the order of steps that have to be followed to extract this information.

Multipath also offers similar functions to extract *DrugBank* drug information and connections to *UniProt* protein entries from parsed *DrugBank* data using the *R* package *dbparser*. Both connections are then integrated and added to a given multilayer graph. It is also important to mention that all 4 types of proteins in *DrugBank* are considered drug targets and extracted to be added to the model. The types are assigned as attributes to the connections. The edges also include the name of the database from which the connection was retrieved, i.e. *DrugBank*, *UniProt*, or both (Figure 22).

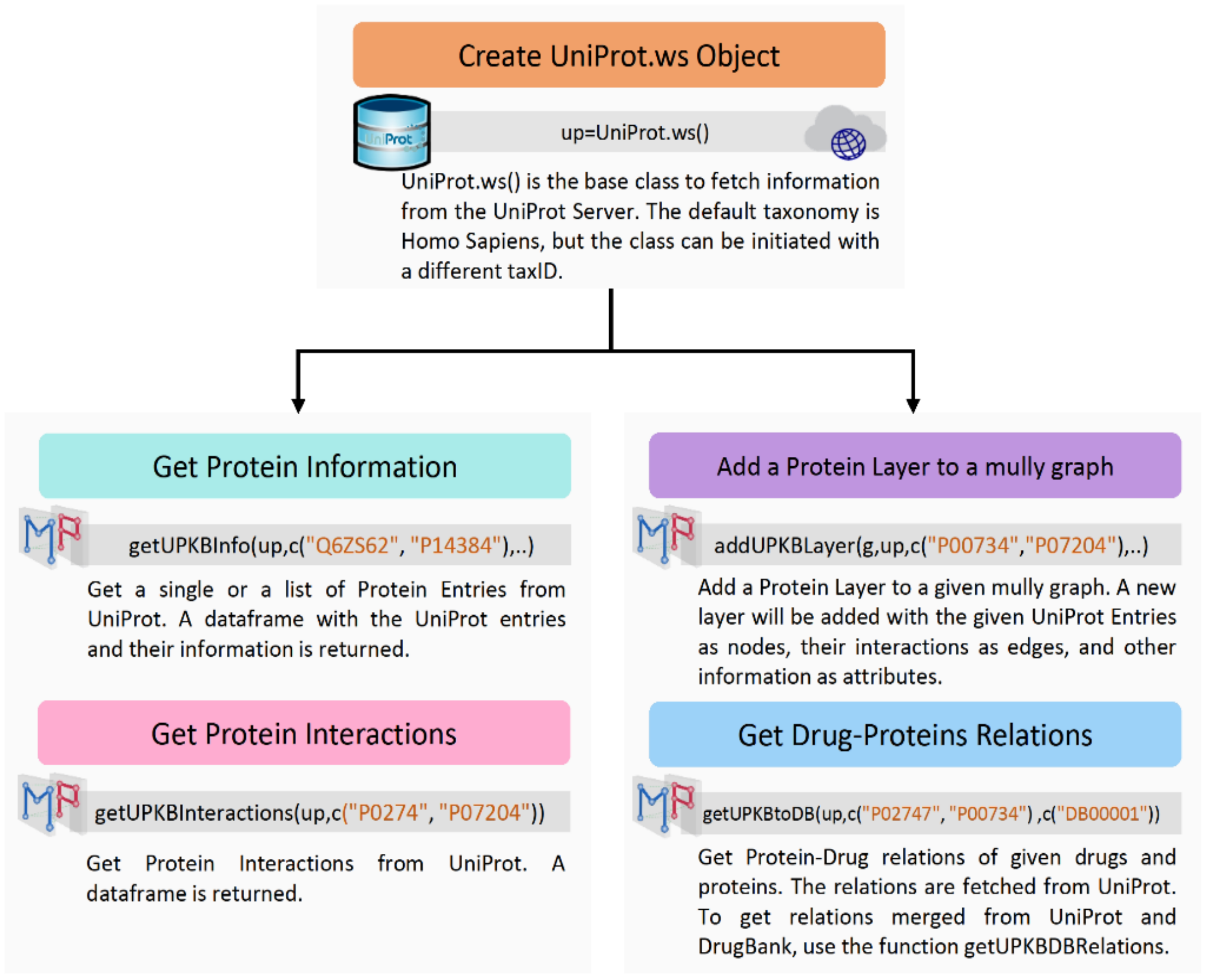

Create UniProt.ws Object
up=UniProt.ws()
UniProt.ws() is the base class to fetch information from the UniProt Server. The default taxonomy is Homo Sapiens, but the class can be initiated with a different taxID.
Get Protein Information
getUPKBInfo(up,c("Q6ZS62", "P14384"),..)
Get a single or a list of Protein Entries from UniProt. A dataframe with the UniProt entries and their information is returned.
Get Protein Interactions
getUPKBInteractions(up,c("P0274", "P07204"))
Get Protein Interactions from UniProt. A dataframe is returned.
Add a Protein Layer to a mully graph
addUPKBLayer(g,up,c("P00734","P07204"),..)
Add a Protein Layer to a given mully graph. A new layer will be added with the given UniProt Entries as nodes, their interactions as edges, and other information as attributes.
Get Drug-Proteins Relations
getUPKBtoDB(up,c("P02747", "P00734") ,c("DB00001"))
Get Protein-Drug relations of given drugs and proteins. The relations are fetched from UniProt. To get relations merged from UniProt and DrugBank, use the function getUPKBDBRelations.

Figure 20 Workflow Diagram to fetch protein data from UniProt.

To demonstrate the usage of the package, a demo code was presented in the paper using the *Signaling by Wnt* pathway. The model was built from *BioPAX* level 3, and the drug information was integrated from *DrugBank* (Figure 21).

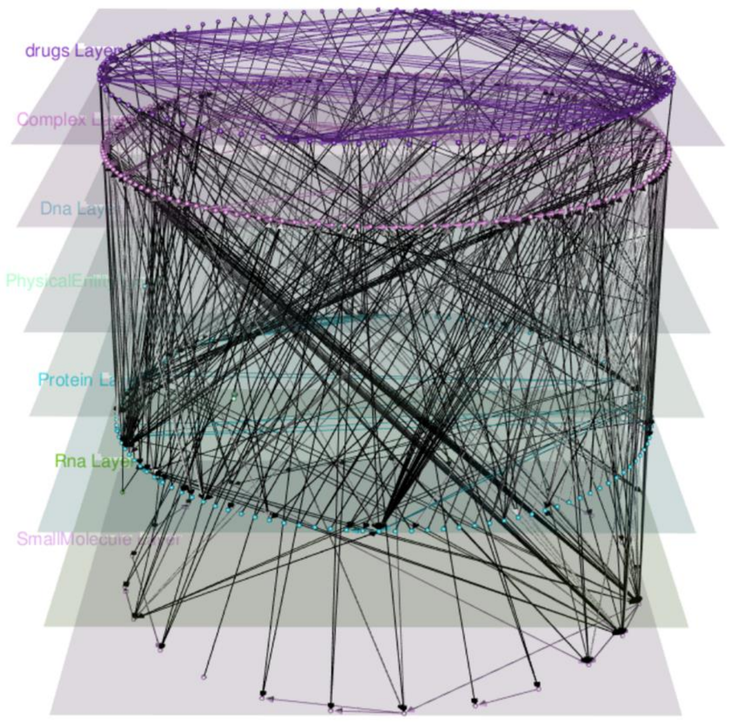
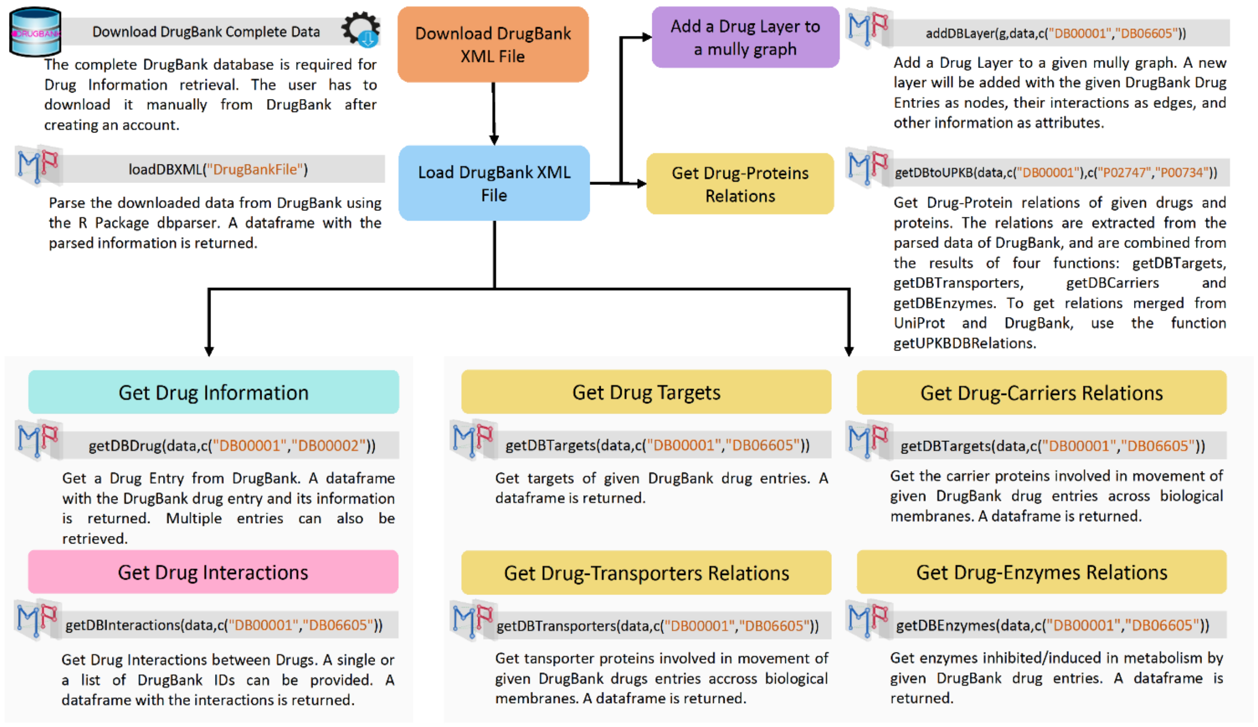


Figure 21 The Signaling by Wnt Multipath Model.

III.3.2. Declaration of my contributions

Prof. Kramer designed the parser called *rBiopaxparser* that I used in this package to parse the *BioPAX*-encoded pathways. I implemented all the functions to get pathways from the *Reactome* database and transform the mono-layered graphs generated by *rBiopaxparser* to *mully* models, as well as the functions to integrate the relevant knowledge from *DrugBank* and *UniProtKB*. I also implemented the *pathwayView* objects, which store the modifications applied to the multilayer pathway models. I used the *Signaling by Wnt* pathway as a demonstration of the usage of the package and applied different changes to the multilayer *Wnt* model. I wrote the original draft of the manuscript and Prof. Kramer approved it.


Download DrugBank XML File
Load DrugBank XML File
Add a Drug Layer to a mully graph
Get Drug-Proteins Relations
Get Drug Information
Get Drug Interactions
Get Drug Targets
Get Drug-Carriers Relations
Get Drug-Transporters Relations
Get Drug-Enzymes Relations

Download DrugBank Complete Data

The complete DrugBank database is required for Drug Information retrieval. The user has to download it manually from DrugBank after creating an account.

```
loadDBXML("DrugBankFile")
```

Parse the downloaded data from DrugBank using the R Package dbparser. A dataframe with the parsed information is returned.

```
addDBLayer(g,data,c("DB00001","DB06605"))
```

Add a Drug Layer to a given mully graph. A new layer will be added with the given DrugBank Drug Entries as nodes, their interactions as edges, and other information as attributes.

```
getDBtoUPKB(data,c("DB00001"),c("P02747","P00734"))
```

Get Drug-Protein relations of given drugs and proteins. The relations are extracted from the parsed data of DrugBank, and are combined from the results of four functions: getDBTargets, getDBTransporters, getDBCarriers and getDBEnzymes. To get relations merged from UniProt and DrugBank, use the function getUPKBDBRelations.

```
getDBDrug(data,c("DB00001","DB00002"))
```

Get a Drug Entry from DrugBank. A dataframe with the DrugBank drug entry and its information is returned. Multiple entries can also be retrieved.

```
getDBInteractions(data,c("DB00001","DB06605"))
```

Get Drug Interactions between Drugs. A single or a list of DrugBank IDs can be provided. A dataframe with the interactions is returned.

```
getDBTargets(data,c("DB00001","DB06605"))
```

Get targets of given DrugBank drug entries. A dataframe is returned.

```
getDBTransporters(data,c("DB00001","DB06605"))
```

Get tansporter proteins involved in movement of given DrugBank drugs entries accross biological membranes. A dataframe is returned.

```
getDBTargets(data,c("DB00001","DB06605"))
```

Get the carrier proteins involved in movement of given DrugBank drug entries across biological membranes. A dataframe is returned.

```
getDBEnzymes(data,c("DB00001","DB06605"))
```

Get enzymes inhibited/induced in metabolism by given DrugBank drug entries. A dataframe is returned.

Figure 22 Workflow Diagram to extract drug data from DrugBank.

III.4. Identification of potential therapeutic targets for host-directed Leishmaniasis treatment and repositioning drugs using network-based approaches

This paper has not been submitted to a journal yet, and the presented version is a ready-to-submit draft.

J. Eduardo Martinez-Hernandez, Zaynab Hammoud, Frank Kramer, Rubens Monte-Neto, Vinicius Maracaja Coutinho, Alberto J. M. Martin*

*Corresponding Author

III.4.1. Summary and discussion

This work was a collaboration between our institute and the *Faculty of Science* at the *Major University* in Santiago, Chile, initiated after J. Eduardo Martinez-Hernandez showed a great interest in the *mully* poster I presented at the *ISMB/ECCB 2019* Conference in Basel, Switzerland. His goal was to find a tool that can integrate omics knowledge from multiple sources into one model. After a discussion about the *mully* package and the future work of *Multipath*, we decided to collaborate on a project that aims to determine which changes in gene regulation of the host (macrophage) occur during the infection, to identify novel, potential drug targets for host-directed *Leishmaniasis* treatment.

Leishmaniasis is a complex of tropical diseases caused by an intracellular parasite called *Leishmania* transmitted by the bite of a sandfly. It is estimated to infect 1.5-2 million new cases with ~70,000 deaths per year, and 350 million prone to infection [62]. The two most known manifestations of *Leishmaniasis* are the *visceral* and *cutaneous* forms, with the latter being neglected by public and private funding organizations due to its low fatality rate [63]. The main symptoms of *Cutaneous Leishmania* are swelling and high temperature. The motivation of this project is the small number of known efficient treatments for this disease, which dispose of many disadvantages, some of which are the high toxicity and costs, and their resistance to the parasite [64].

The analysis started with the publicly available *RNA-seq* data derived from Leishmania major infected macrophage, which comprises four different time points, namely at 4, 24, 48, and 72 hours post-infection (hpi). The tool *FastQC* was used to apply quality control and eliminate low-quality reads, followed by the *HTSeq-count* 0.7.2 to calculate the gene expression values. After normalization and testing, differentially expressed genes with a p-value ≤ 0.5 and $|\log \text{fold change}| \geq 0.5$ were filtered. The final list of potential genes in interest was obtained by selecting only immune response, stress response, or host-pathogen interaction related genes based on the *GO* enrichment analysis. The genes were selected from the filtered gene regulatory network of *DoRothEA* using the *RNA-seq* data, and comparing infected and non-infected models at the different time points (Figure 23).

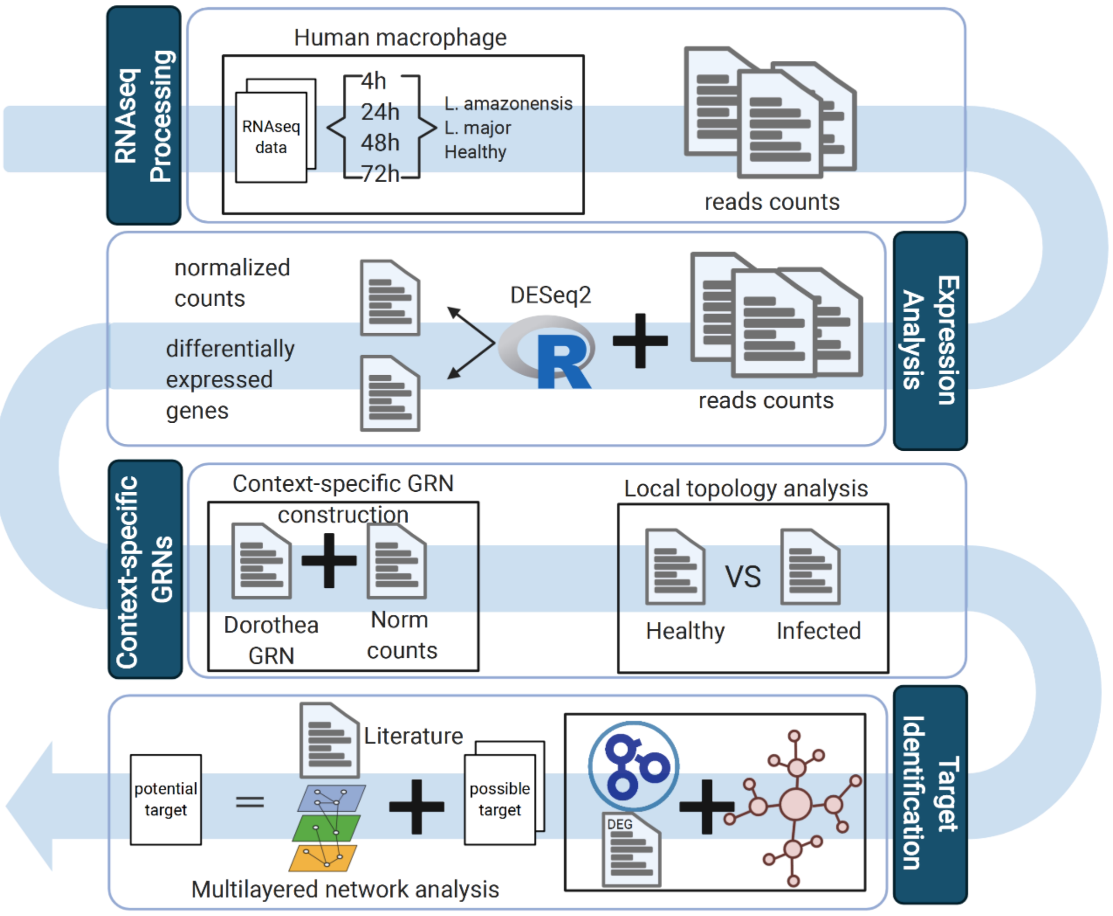


Figure 23 Pipeline to identify potential therapeutic target for Leishmaniasis treatment in macrophage from RNA-seq data.

This resulted in a list of 113 genes, of which the corresponding *HGNC* symbols were used to obtain *UniProt* IDs of their gene products (Figure 24). This mapping was conducted using the *UniProt.ws* package, followed by a mapping of the resulted 909 unique *UniProt* IDs to 313 pathways in the *Reactome* database. Using the package *Multipath*, the pathways' *BioPAX* level 3 files were downloaded and modelled as *mully* graphs. The proteins obtained in our gene-mapping were mapped to their internal IDs in each graph and filtered by deleting all non-protein layers (Complex, Small Molecules, RNA …), followed by all non-mapped protein nodes. The drug targets were then extracted from the downloaded *DrugBank* data, parsed using the package *dbparser* and added as new layers to each *mully* object. This addition includes drug-drug interactions, which can also play a major role in the prediction of potential drug targets. The interactions between the drugs and proteins were combined from *UniProt* and *DrugBank*. It is also important to mention that the drug targets obtained from *DrugBank* are of four types (see Chapter DrugBank). The additional information to each drug entry was also extracted, such as the group (approved, investigational, withdrawn, vet-approved, etc.), names, type (biotech, small molecule, etc.), state (liquid, solid), etc.

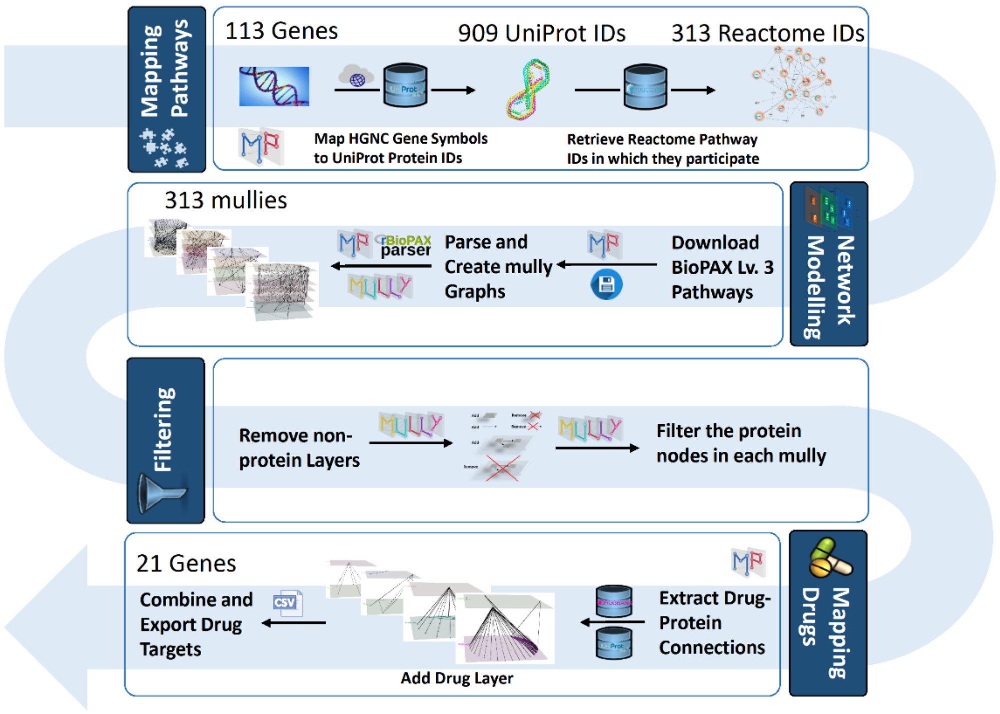

Mapping Pathways
113 Genes
909 UniProt IDs
313 Reactome IDs
Map HGNC Gene Symbols to UniProt Protein IDs
Retrieve Reactome Pathway IDs in which they participate
Network Modelling
313 mullies
BioPAX parser
Parse and Create mully Graphs
Download BioPAX Lv. 3 Pathways
Filtering
Remove non-protein Layers
Filter the protein nodes in each mully
Mapping Drugs
21 Genes
Combine and Export Drug Targets
Extract Drug-Protein Connections
Add Drug Layer

Figure 24 The workflow Diagram to identify the list of drug targets.

III.4.2. Declaration of my contributions

J. Eduardo Martinez-Hernandez realised the pipeline to identify the list of genes of interest. He conducted the gene expression analysis on public *RNA-seq* data, alongside the *GO* enrichment analysis. He compared regulatory networks between infected and non-infected, looking for the difference, and recovering the nodes or the genes only present in the infected macrophage, and related to an immune response I received a list of *HGNC* symbols of the genes, which I mapped to gene products from *UniProt*, in order to identify the corresponding pathways. I used the packages that I implemented to fetch and map the data, build the multilayer graphs from the obtained pathways, and filter the results to identify the drug targets. The additional information on the obtained list of drugs was fetched from *DrugBank*, and the result was sent to Eduardo Martinez, who studied and researched each single drug target on the list to determine which drug targets present the highest possibility to be proposed as a treatment. Rubens Monte-Neto, a member of the laboratory team in Brazil also carried a set of experiments, in order to select the set of drugs to check if they have a therapeutic effect, by infecting some macrophage with the parasite and injecting them with the drugs to observe the immunity response. The paper was written by Eduardo Martinez and me, and my part was revised by Prof. Kramer.

IV. Summary and Outlook

In this dissertation, four publications were presented. In the first publication, thorough literature research was conducted to undermine the concept of multilayer networks. I presented the different mathematical formulations of multilayer graphs, starting from a generalized classification into two abstract categories, being node- and edge-coloured. I described the difference between them and showed the different practical implementations, ranging from *Multiplex* to *Interconnected*, to *Hypergraphs*. Most importantly, I summarized the research that has been conducted employing a multilayer representation, and I categorized each one into the aforementioned aspects. This helped me project previous implementations and employments of multilayer graphs into my project in order to decide which implementation and aspect advantageously serve my goals.

In the second paper, I present my *mully R* package as a realisation of this literature research. The package was implemented independently in order to address multiple users. It allows the creation, modification, and visualization of node-coloured multilayer graphs. It offers a set of standard procedures, such as the addition and deletion of elements, as well as more advanced features, such as 3D visualization, transitivity, and merging. The package was updated multiple times in the past years and was recently uploaded to the *CRAN* repository. The package has received remarkable attention from many scientists and researchers in the different conferences that I attended, and is being used by many users. This attention led me particularly to meet the researcher, with whom I collaborated to produce and realize the work of the fourth publication.

The *R* package *Multipath* was presented in the third publication, being the heart of my Ph.D. The different publications presented so far were small puzzle pieces to achieve a bigger milestone being this package. In this package, I implemented functions based on state-of-the-art tools and packages to transform *BioPAX*-encoded pathways into multilayer graphs. The outstanding feature of these models is their reproducible and integrated nature. *Multipath* allows generating multiple views of a given mully pathway model, while documenting all applied changes, thus helping to reproduce the original or previous versions of the model. Furthermore, it comprises functions that retrieve additional related and relevant information on the nodes in the graph. The supported databases are currently *Reactome* for pathways, *UniProtKB* for proteins, and *DrugBank* for drugs.

The last article is the result of a fruitful collaboration between my institute and the Major University in Santiago, Chile. Its aim was to identify potential drug targets in the host-directed treatment of a neglected disease called *Cutaneous Leishmania*. The main role that I had in this collaboration was to extract drug relations of a given set of genes. To do so, the package *Multipath* was used to map the genes to their gene products in *UniProt*, hence allowing the retrieval of the corresponding pathways in which they play a role. The identified pathways were modelled and the relevant drug layers were added. Filtering was required to obtain the final list of potential drugs that were researched by my collaborator. The drugs are also being tested *in vitro* and the biological results will be published soon.

Although I believe my work contributes generously in addressing big challenges and problems faced in the field of *Bioinformatics*, I am also certain that many more contributions are still needed. This work will not be terminated with the end of my Ph.D.

I am currently working on further evaluating my packages, using *TCGA* cancer data. Specifically, I am interested in the *HTSeq-FPKM* gene expression data of the breast cancer studies *brca* of the *TCGA* project. The models of the pathways involving the genes *BRCA1* and *BRCA2* are modelled using *Multipath*. The patients in the studies will be added as nodes to an additional layer. Furthermore, the models will be annotated accordingly using the gene expression data, by connecting the patients to the corresponding genes or proteins.

I am continuously working to maintain, improve and update both of my packages with new findings and better frameworks. The package *mully* will be updated to support more concrete implementations of the multilayer networks that I presented. I am also planning on allowing multiple aspects of one network in the same *mully* graph, for instance, merging node- and edge-coloured aspects. More specifically, gene regulatory networks created and extracted from the parsed pathways can be also modelled as edge-coloured graphs, by integrating co-expression data. The model can then be merged with the original pathway, by adding the multiple gene layers. Regarding the package *Multipath*, and as mentioned in the publication, the package is to be updated with version 2.0 supporting more databases to be integrated into the same model. For instance, a great addition would be *ChEBI* information added to the nodes on the small molecule layer, and *Ensembl* [26] for the gene information.

Another related work is the implementation of a *Reactome* online catalog that is planned to be available by the end of September of 2021. The catalog will contain generated *mully* models of all available pathways in *Reactome*, ready to be fetched from a *Neo4j* Graph Database. The ready models are planned to include a considerable number of relevant information.

Finally, the main result that I obtained from this dissertation is that the field of knowledge has no limit. Although I believe that seeking perfection in every task is a necessity, it can never in the least be reached easily. Research requires a great deal of creativity and innovation in order to contribute adequately to the finding of new solutions. The greatest lesson that I learned is that every solution solves a problem, and reveals many more. This is the heart and soul of research, which allows continuous birth cycles of new ideas and theories.

V. References

1. Junker, B.H. (2008). Networks in Biology. In Analysis of Biological Networks (eds Y. Pan, A.Y. Zomaya, B.H. Junker and F. Schreiber). https://doi.org/10.1002/9780470253489.ch1
2. Andreopoulos W, Labudde D. Protein-protein interaction networks. 2013. http://www.bioforscher.de/bigM/ippb9076rp8sityx/manager/documents/general/pdf/books/chapters/protein_protein_interaction_networks.pdf Accessed 2 Dec 2019.
3. Safari-Alighiarloo N, Taghizadeh M, Rezaei-Tavirani M, Goliaei B, Peyvandi AA. Protein-protein interaction networks (PPI) and complex diseases. Gastroenterol Hepatol Bed Bench. 2014;7(1):17–31.
4. Aittokallio T, Schwikowski B. Graph-based methods for analysing networks in cell biology. Brief Bioinform. 2006 Sep 1;7(3):243–55. https://doi.org/10.1093/bib/bbl022
5. Saraiya P, North C, Duca K. Visualizing Biological Pathways: Requirements Analysis, Systems Evaluation and Research Agenda. Information Visualization. 2005 Sep 1;4(3):191–205. https://doi.org/10.1057/palgrave.ivs.9500102
6. Vyas H, Summers R. Interoperability of bioinformatics resources. VINE. 2005 Jan 1;35(3):132–9. https://doi.org/10.1093/bib/bbs074
7. Kwiatkowska MZ, Heath JK. Biological pathways as communicating computer systems. Journal of Cell Science. 2009 Aug 15;122(16):2793–800. https://doi.org/10.1242/jcs.039701
8. Hothorn T, Leisch F. Case studies in reproducibility. Briefings in Bioinformatics. 2011 May 1;12(3):288–300. https://doi.org/10.1093/bib/bbq084
9. Kim Y-M, Poline J-B, Dumas G. Experimenting with reproducibility: a case study of robustness in bioinformatics. GigaScience [Internet]. 2018 Jul 1 [cited 2021 Jan 27];7(giy077). https://doi.org/10.1093/gigascience/giy077
10. Demir, E., Cary, M., Paley, S. et al. The BioPAX community standard for pathway data sharing. Nat Biotechnol 28, 935–942 (2010). https://doi.org/10.1038/nbt.1666
11. Hothorn T, Held L, Friede T. Biometrical Journal and Reproducible Research. Biometrical Journal. 2009;51(4):553–5. https://doi.org/10.1002/bimj.200900154
12. Narsingh Deo. 1974. Graph Theory with Applications to Engineering and Computer Science (Prentice Hall Series in Automatic Computation). Prentice-Hall, Inc., USA.
13. Xu J. Theory and Application of Graphs. Springer Science & Business Media; 2003. 346 p.
14. John Adrian Bondy. 1976. Graph Theory With Applications. Elsevier Science Ltd., GBR.
15. Allard A, Noël P-A, Dubé LJ, Pourbohloul B. Heterogeneous bond percolation on multitype networks with an application to epidemic dynamics. Phys Rev E. 2009 Mar 26;79(3):036113. https://doi.org/10.1103/PhysRevE.79.036113
16. Hammoud Z, Kramer F. Multilayer networks: aspects, implementations, and application in biomedicine. Big Data Analytics. 2020 Jul 6;5(1):2. https://doi.org/10.1186/s41044-020-00046-0
17. Cary MP, Bader GD, Sander C. Pathway information for systems biology. FEBS Letters. 2005 Mar 21;579(8):1815–20. https://doi.org/10.1016/j.febslet.2005.02.005
18. Pavlopoulos GA, Secrier M, Moschopoulos CN, Soldatos TG, Kossida S, Aerts J, et al. Using graph theory to analyze biological networks. BioData Mining. 2011 Apr 28;4(1):10. https://doi.org/10.1186/1756-0381-4-10
19. Pico AR, Kelder T, van Iersel MP, Hanspers K, Conklin BR, et al. (2008) WikiPathways: Pathway Editing for the People. PLOS Biology 6(7): e184. https://doi.org/10.1371/journal.pbio.0060184
20. Gary D. Bader, Michael P. Cary, Chris Sander, Pathguide: a Pathway Resource List, Nucleic Acids Research, Volume 34, Issue suppl_1, 1 January 2006, Pages D504–D506, https://doi.org/10.1093/nar/gkj126
21. Igor Rodchenkov, Ozgun Babur, Augustin Luna, Bulent Arman Aksoy, Jeffrey V Wong, Dylan Fong, Max Franz, Metin Can Siper, Manfred Cheung, Michael Wrana, Harsh Mistry, Logan Mosier, Jonah Dlin, Qizhi Wen, Caitlin O'Callaghan, Wanxin Li, Geoffrey Elder, Peter T Smith,

Christian Dallago, Ethan Cerami, Benjamin Gross, Ugur Dogrusoz, Emek Demir, Gary D Bader, Chris Sander, Pathway Commons 2019 Update: integration, analysis and exploration of pathway data, Nucleic Acids Research, Volume 48, Issue D1, 08 January 2020, Pages D489–D497, https://doi.org/10.1093/nar/gkz946

22. Minoru Kanehisa, Michihiro Araki, Susumu Goto, Masahiro Hattori, Mika Hirakawa, Masumi Itoh, Toshiaki Katayama, Shuichi Kawashima, Shujiro Okuda, Toshiaki Tokimatsu, Yoshihiro Yamanishi, KEGG for linking genomes to life and the environment, Nucleic Acids Research, Volume 36, Issue suppl_1, 1 January 2008, Pages D480–D484, https://doi.org/10.1093/nar/gkm882
23. Minoru Kanehisa, Susumu Goto, Miho Furumichi, Mao Tanabe, Mika Hirakawa, KEGG for representation and analysis of molecular networks involving diseases and drugs, Nucleic Acids Research, Volume 38, Issue suppl_1, 1 January 2010, Pages D355–D360, https://doi.org/10.1093/nar/gkp896
24. Mathias Krull, Nico Voss, Claudia Choi, Susanne Pistor, Anatolij Potapov, Edgar Wingender, TRANSPATH ® : an integrated database on signal transduction and a tool for array analysis , Nucleic Acids Research, Volume 31, Issue 1, 1 January 2003, Pages 97–100, https://doi.org/10.1093/nar/gkg089
25. Mathias Krull, Susanne Pistor, Nico Voss, Alexander Kel, Ingmar Reuter, Deborah Kronenberg, Holger Michael, Knut Schwarzer, Anatolij Potapov, Claudia Choi, Olga Kel-Margoulis, Edgar Wingender, TRANSPATH®: an information resource for storing and visualizing signaling pathways and their pathological aberrations, Nucleic Acids Research, Volume 34, Issue suppl_1, 1 January 2006, Pages D546–D551, https://doi.org/10.1093/nar/gkj107
26. T. Hubbard, D. Andrews, M. Caccamo, G. Cameron, Y. Chen, M. Clamp, L. Clarke, G. Coates, T. Cox, F. Cunningham, V. Curwen, T. Cutts, T. Down, R. Durbin, X. M. Fernandez-Suarez, J. Gilbert, M. Hammond, J. Herrero, H. Hotz, K. Howe, V. Iyer, K. Jekosch, A. Kahari, A. Kasprzyk, D. Keefe, S. Keenan, F. Kokocinsci, D. London, I. Longden, G. McVicker, C. Melsopp, P. Meidl, S. Potter, G. Proctor, M. Rae, D. Rios, M. Schuster, S. Searle, J. Severin, G. Slater, D. Smedley, J. Smith, W. Spooner, A. Stabenau, J. Stalker, R. Storey, S. Trevanion, A. Ureta-Vidal, J. Vogel, S. White, C. Woodwark, E. Birney, Ensembl 2005, Nucleic Acids Research, Volume 33, Issue suppl_1, 1 January 2005, Pages D447–D453, https://doi.org/10.1093/nar/gki138
27. Nuala A. O'Leary, Mathew W. Wright, J. Rodney Brister, Stacy Ciufo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, Alexander Astashyn, Azat Badretdin, Yiming Bao, Olga Blinkova, Vyacheslav Brover, Vyacheslav Chetvernin, Jinna Choi, Eric Cox, Olga Ermolaeva, Catherine M. Farrell, Tamara Goldfarb, Tripti Gupta, Daniel Haft, Eneida Hatcher, Wratko Hlavina, Vinita S. Joardar, Vamsi K. Kodali, Wenjun Li, Donna Maglott, Patrick Masterson, Kelly M. McGarvey, Michael R. Murphy, Kathleen O'Neill, Shashikant Pujar, Sanjida H. Rangwala, Daniel Rausch, Lillian D. Riddick, Conrad Schoch, Andrei Shkeda, Susan S. Storz, Hanzhen Sun, Francoise Thibaud-Nissen, Igor Tolstoy, Raymond E. Tully, Anjana R. Vatsan, Craig Wallin, David Webb, Wendy Wu, Melissa J. Landrum, Avi Kimchi, Tatiana Tatusova, Michael DiCuccio, Paul Kitts, Terence D. Murphy, Kim D. Pruitt, Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation, Nucleic Acids Research, Volume 44, Issue D1, 4 January 2016, Pages D733–D745, https://doi.org/10.1093/nar/gkv1189
28. The UniProt Consortium, UniProt: a worldwide hub of protein knowledge, Nucleic Acids Research, Volume 47, Issue D1, 08 January 2019, Pages D506–D515, https://doi.org/10.1093/nar/gky1049
29. Hester M. Wain, Michael J. Lush, Fabrice Ducluzeau, Varsha K. Khodiyar, Sue Povey, Genew: the Human Gene Nomenclature Database, 2004 updates, Nucleic Acids Research, Volume 32, Issue suppl_1, 1 January 2004, Pages D255–D257, https://doi.org/10.1093/nar/gkh072
30. Harris MA, Clark J, Ireland A, Lomax J, Ashburner M, Foulger R, Eilbeck K, Lewis S, Marshall B, Mungall C, Richter J, Rubin GM, Blake JA, Bult C, Dolan M, Drabkin H, Eppig JT, Hill DP, Ni L, Ringwald M, Balakrishnan R, Cherry JM, Christie KR, Costanzo MC, Dwight SS, Engel S, Fisk DG, Hirschman JE, Hong EL, Nash RS, Sethuraman A, Theesfeld CL, Botstein D, Dolinski K,

Feierbach B, Berardini T, Mundodi S, Rhee SY, Apweiler R, Barrell D, Camon E, Dimmer E, Lee V, Chisholm R, Gaudet P, Kibbe W, Kishore R, Schwarz EM, Sternberg P, Gwinn M, Hannick L, Wortman J, Berriman M, Wood V, de la Cruz N, Tonellato P, Jaiswal P, Seigfried T, White R; Gene Ontology Consortium. The Gene Ontology (GO) database and informatics resource. Nucleic Acids Res. 2004 Jan 1;32(Database issue):D258-61. https://doi.org/10.1093/nar/gkh036

31. T. Heinemeyer, X. Chen, H. Karas, A. E. Kel, O. V. Kel, I. Liebich, T. Meinhardt, I. Reuter, F. Schacherer, E. Wingender, Expanding the TRANSFAC database towards an expert system of regulatory molecular mechanisms, Nucleic Acids Research, Volume 27, No. 1, 1999.
32. The TRANSPATH Website. http://genexplain.com/transpath/. [Accessed on 29.01.2021]
33. Livigni, A., O'Hara, L., Polak, M. et al. A graphical and computational modeling platform for biological pathways. Nat Protoc 13, 705–722 (2018). https://doi.org/10.1038/nprot.2017.144
34. Novère, N., Hucka, M., Mi, H. et al. The Systems Biology Graphical Notation. Nat Biotechnol 27, 735–741 (2009). https://doi.org/10.1038/nbt.1558
35. Smith, L., Wilkinson, D., Hucka, M. et al. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. Nat Prec (2010). https://doi.org/10.1038/npre.2010.4959.1
36. Deutsch EW, Orchard S, Binz PA, Bittremieux W, Eisenacher M, Hermjakob H, Kawano S, Lam H, Mayer G, Menschaert G, Perez-Riverol Y, Salek RM, Tabb DL, Tenzer S, Vizcaíno JA, Walzer M, Jones AR. Proteomics Standards Initiative: Fifteen Years of Progress and Future Work. J Proteome Res. 2017 Dec 1;16(12):4288-4298. https://doi.org/10.1021/acs.jproteome.7b00370
37. Lloyd CM, Halstead MDB, Nielsen PF. CellML: its future, present and past. Progress in Biophysics and Molecular Biology. 2004 Jun 1;85(2):433–50. https://doi.org/10.1016/j.pbiomolbio.2004.01.004
38. Yuh-Mei Liao, Hamid Ghanadan. The Chemical Markup Language. Anal. Chem. 2002, 74, 13, 389 A–390 A. https://doi.org/10.1021/ac0220676
39. Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res. 2003 Nov;13(11):2498-504. https://doi.org/10.1101/gr.1239303
40. Demir E, Babur Ö, Rodchenkov I, Aksoy BA, Fukuda KI, et al. (2013) Using Biological Pathway Data with Paxtools. PLOS Computational Biology 9(9): e1003194. https://doi.org/10.1371/journal.pcbi.1003194
41. The DrugBank Website. https://go.drugbank.com/. [Accessed on 29.01.2021]
42. Wishart DS, Knox C, Guo AC, Shrivastava S, Hassanali M, Stothard P, et al. DrugBank: a comprehensive resource for in silico drug discovery and exploration. Nucleic Acids Res. 2006 Jan 1;34(Database issue):D668–72. https://doi.org/10.1093/nar/gkj067
43. Wishart DS, Knox C, Guo AC, Cheng D, Shrivastava S, Tzur D, et al. DrugBank: a knowledgebase for drugs, drug actions and drug targets. Nucleic Acids Res. 2008 Jan;36(Database issue):D901-906. https://doi.org/10.1093/nar/gkm958
44. Knox C, Law V, Jewison T, Liu P, Ly S, Frolkis A, et al. DrugBank 3.0: a comprehensive resource for 'Omics' research on drugs. Nucleic Acids Res. 2011 Jan;39(Database issue):D1035–41. https://doi.org/10.1093/nar/gkq1126
45. Law V, Knox C, Djoumbou Y, Jewison T, Guo AC, Liu Y, et al. DrugBank 4.0: shedding new light on drug metabolism. Nucleic Acids Res. 2014 Jan 1;42(Database issue):D1091–7. https://doi.org/10.1093/nar/gkt1068
46. Wishart DS, Feunang YD, Guo AC, Lo EJ, Marcu A, Grant JR, et al. DrugBank 5.0: a major update to the DrugBank database for 2018. Nucleic Acids Res. 2018 04;46(D1):D1074–82. https://doi.org/10.1093/nar/gkx1037
47. Baris E. Suzek, Hongzhan Huang, Peter McGarvey, Raja Mazumder, Cathy H. Wu, UniRef: comprehensive and non-redundant UniProt reference clusters, Bioinformatics, Volume 23, Issue 10, 15 May 2007, Pages 1282–1288, https://doi.org/10.1093/bioinformatics/btm098
48. Baris E. Suzek, Yuqi Wang, Hongzhan Huang, Peter B. McGarvey, Cathy H. Wu, the UniProt Consortium, UniRef clusters: a comprehensive and scalable alternative for improving sequence

similarity searches, Bioinformatics, Volume 31, Issue 6, 15 March 2015, Pages 926–932, https://doi.org/10.1093/bioinformatics/btu739

49. Rasko Leinonen, Federico Garcia Diez, David Binns, Wolfgang Fleischmann, Rodrigo Lopez, Rolf Apweiler, UniProt archive, Bioinformatics, Volume 20, Issue 17, 22 November 2004, Pages 3236–3237, https://doi.org/10.1093/bioinformatics/bth191
50. G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G.R. Gopinath, G.R. Wu, L. Matthews, S. Lewis, E. Birney, L. Stein, Reactome: a knowledgebase of biological pathways, Nucleic Acids Research, Volume 33, Issue suppl_1, 1 January 2005, Pages D428–D432, https://doi.org/10.1093/nar/gki072
51. Kirill Degtyarenko, Paula de Matos, Marcus Ennis, Janna Hastings, Martin Zbinden, Alan McNaught, Rafael Alcántara, Michael Darsow, Mickaël Guedj, Michael Ashburner, ChEBI: a database and ontology for chemical entities of biological interest, Nucleic Acids Research, Volume 36, Issue suppl_1, 1 January 2008, Pages D344–D350, https://doi.org/10.1093/nar/gkm791
52. Antonio Fabregat, Steven Jupe, Lisa Matthews, Konstantinos Sidiropoulos, Marc Gillespie, Phani Garapati, Robin Haw, Bijay Jassal, Florian Korninger, Bruce May, Marija Milacic, Corina Duenas Roca, Karen Rothfels, Cristoffer Sevilla, Veronica Shamovsky, Solomon Shorser, Thawfeek Varusai, Guilherme Viteri, Joel Weiser, Guanming Wu, Lincoln Stein, Henning Hermjakob, Peter D’Eustachio, The Reactome Pathway Knowledgebase, Nucleic Acids Research, Volume 46, Issue D1, 4 January 2018, Pages D649–D655, https://doi.org/10.1093/nar/gkx1132
53. David Croft, Antonio Fabregat Mundo, Robin Haw, Marija Milacic, Joel Weiser, Guanming Wu, Michael Caudy, Phani Garapati, Marc Gillespie, Maulik R. Kamdar, Bijay Jassal, Steven Jupe, Lisa Matthews, Bruce May, Stanislav Palatnik, Karen Rothfels, Veronica Shamovsky, Heeyeon Song, Mark Williams, Ewan Birney, Henning Hermjakob, Lincoln Stein, Peter D'Eustachio, The Reactome pathway knowledgebase, Nucleic Acids Research, Volume 42, Issue D1, 1 January 2014, Pages D472–D477, https://doi.org/10.1093/nar/gkt1102
54. The Neo4j Website. https://neo4j.com/. [Accessed on 29.01.2021]
55. Fabregat, A., Sidiropoulos, K., Viteri, G. et al. Reactome pathway analysis: a high-performance in-memory approach. BMC Bioinformatics 18, 142 (2017). https://doi.org/10.1186/s12859-017-1559-2
56. The Reactome Website. https://reactome.org/. [Accessed on 29.01.2021]
57. Robert Kabacoff. 2015. R in Action: Data Analysis and Graphics with R. Manning Publications Co., USA. ISBN:978-1-61729-138-8.
58. Norman Matloff. 2011. The Art of R Programming: A Tour of Statistical Software Design (1st. ed.). No Starch Press, USA. ISBN:978-1-59327-384-2.
59. Joseph Adler. 2012. R in a Nutshell. O'Reilly Media, Inc. ISBN:978-1-4493-1208-4.
60. Kramer F, Bayerlová M, Klemm F, Bleckmann A, Beissbarth T. rBiopaxParser--an R package to parse, modify and visualize BioPAX data. Bioinformatics. 2013 Feb 15;29(4):520–2. https://doi.org/10.1093/bioinformatics/bts710
61. The igraph R package. https://igraph.org/r/. [Accessed on 29.01.2021]
62. Torres-Guerrero, E., Quintanilla-Cedillo, M. R., Ruiz-Esmenjaud, J., & Arenas, R. (2017). Leishmaniasis: a review. F1000Research, 6, 750. https://doi.org/10.12688/f1000research.11120.1
63. Reithinger R, Dujardin JC, Louzir H, Pirmez C, Alexander B, Brooker S. Cutaneous leishmaniasis. Lancet Infect Dis. 2007 Sep;7(9):581-96. https://doi/10.1016/S1473-3099(07)70209-8
64. Matos APS, Viçosa AL, Ré MI, Ricci-Júnior E, Holandino C. A review of current treatments strategies based on paromomycin for leishmaniasis. Journal of Drug Delivery Science and Technology. 2020 Jun 1;57:101664. https://doi.org/10.1016/j.jddst.2020.101664

VI. Appendix

Appendix A

Publication “Multilayered Networks: Aspects, Implementations, and Application in Biomedicine”

Big Data Analytics

REVIEW Open Access

Multilayer networks: aspects, implementations, and application in biomedicine

Zaynab Hammoud[*] and Frank Kramer

* Correspondence:
zaynabhassanhammoud@gmail.com
IT-Infrastructure for Translational Medical Research, University of Augsburg, Augsburg, Germany

Abstract

Modeling and analyses of complex systems using network theory have been an object of study for a long time. They have caught attention in many disciplines such as sociology, epidemiology, ecology, psychology, biology, biomedicine, and other fields. Network theory is especially an efficient tool to model biological networks such as gene co-expression networks, protein-protein interaction networks, or pathways. Considering the enhanced resolutions of complex real-world systems, the interest has been directed to multilayered networks. However, despite this surge of recent attention, the use of the multilayer framework in the biological field is still in its youth. In this paper, we review the different aspects and terminologies of multilayered networks. We also briefly discuss the variant applications of the multilayer framework, and finally, we give an overview of various existing applications of the multilayer model in network biology.

Keywords: Multilayered graphs, Graph theory, Network biology

Background

In this review, we address the lack of a terminology convention in the field of multilayered networks, by distinguishing their different abstract formulation and aspects. For each aspect, we distinguish the appropriate applications, based on its structure and the state-of-the-art articles. We also describe and illustrate the different implementations of these aspects, with their assets and usage in previous research. In addition, we enumerate the various published state-of-the-art articles, in which the authors used a multilayer framework to model networks in Biomedicine, and solve different biomedical problems. We discuss the convenience of this usage and the reasons to choose specific implementations to solve specific problems.

Introduction

Network theory has been used for many years in the modeling and analysis of complex systems. As the data evolves and becomes more heterogeneous and complex,

© The Author(s). 2020 **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/. The Creative Commons Public Domain Dedication waiver (http://creativecommons.org/publicdomain/zero/1.0/) applies to the data made available in this article, unless otherwise stated in a credit line to the data.

monoplex networks become an oversimplification of the corresponding systems [1, 2]. In the case of biological networks, the represented systems are complex as well as the problems studied, and exploiting information from a new perspective might improve the understanding of biological structure and functionality [3]. In fact, the classical network representation used for years seems to fail against the heterogeneity of the objects and the relations. This imposes a need to go beyond traditional networks into a richer framework capable of hosting objects and relations of different scales [4], called Multilayer Network.

The term Multilayer Networks has been used by many scientists over the years. These networks have many definitions and implementations. One of these definitions is presented by Allard et al. [5], who define a Multilayer or a Multitype Network by a network with a set of N Nodes each assigned a type from a set of M types. However, the definition of a Multilayer Network is much broader. The simplest definition of a multilayer network is a set of nodes, edges, and layers, where the interpretation of the layers depends on the implementation of the model. One of the main problems faced when studying these graphs is the absence of a terminology and a nomenclature convention. In their review of Multilayer Networks, Kivelä et al. [4] summarize the different naming of these networks in a table of 26 different names. We have come to learn that some of the names are representing the same structure and framework, but others depend on the type of the graph.

Multilayer Networks provide better modeling for complex networks, like biological networks. These networks are heterogeneous in different manners since any biological function is rarely depending on just one element or type of interaction between the elements. The importance of their usage in the domain of biomedicine was briefly discussed by some researchers [1], but to our knowledge, it was never assimilated and reviewed. With this review, we aim at summarizing the different state-of-the-art articles that used a multilayer framework in biomedicine. This article is divided into 3 sections. In the first one, we present an abstract formulation of a multilayer graph and the different criteria for choosing layers. We explain the transformation from a monolayer to a multilayer framework. The second section explores the implementation of the multilayer framework, as well as usage examples. In the third section, we summarize some of the state-of-the-art articles that have been published in different fields of biomedicine and are all using one of the different implementations of multilayer networks.

Multilayered networks: aspects and terminologies

The multilayer nature of graphs has been a subject of study for multiple years. Considering that this framework has many applications in various fields, its interpretation and implementation depend on the subject it is serving. The main difference between the types is the criteria to link a node to a layer. It is indeed crucial to determine how the layers should be assigned when converting from a monoplex to a multilayer model [4]. In this context, two types of networks can be distinguished.

Node-colored graphs

Node-colored graphs (Fig. 1) are representations of systems with heterogeneous nodes. Nodes have different aspects or types, defined by colors. The graph G_N (Eq. 1) is a tuple

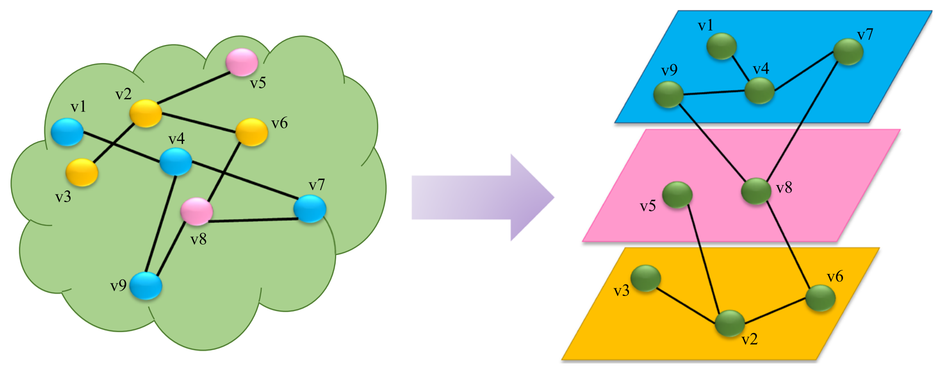


Fig. 1 Node-colored Graphs. Nodes having similar colors are grouped in the same layer. The edges are the same as in the original network

defined by a set of nodes V, a set of edges E, and a set of layers C, where layers represent the color of the nodes [4]. Two nodes on layers α and β are connected with an edge $E_{\alpha\beta}$.

$$\mathrm{G_N} = (V, E, C),\ \ \mathrm{V}_\alpha \in V \times C,\ \ \mathrm{E}_{\alpha\beta} \in V_\alpha \times V_\beta,\ \ where\ \ \alpha, \beta \in C \tag{1}$$

In such graphs, the focus is exploiting the topology of the system from a node-based perspective. In most of the cases, the graph is layer-disjoint, i.e. the node can only belong to one layer, having only one color.

Edge-colored graphs

Edge-colored graphs (Fig. 2) are representations of systems with heterogeneous connections between the nodes. The graph $\mathrm{G_E}$ (Eq. 2) is a tuple defined by a set of nodes V, a set of colors C with length b, and a set of edges E, each assigned a type or aspect defined by the color. Edges of the same color belong to the same layer. Each layer can contain a subset or the complete set of nodes.

$$\begin{gathered} G_E = (V, E, C) \\ \{G_\alpha\}_{\alpha=1}^{b} = \{(V_\alpha, E_\alpha)\}_{\alpha=1}^{b}, \forall \alpha, \beta \in C \quad \mathrm{V}_{\alpha =} V_\beta, \\ \mathrm{E}_\alpha \in V_\alpha \times V_\alpha \times C \\ \{G_\alpha\}_{\alpha=1}^{b} = \{(V_\alpha, E_\alpha)\}_{\alpha=1}^{b} \end{gathered} \tag{2}$$

The graph is node-aligned, i.e. the same node can belong to different layers simultaneously. In such graphs, the focus is on the dynamics of the network and the interactions between its components. It is very efficient to use when studying different aspects of the same system. Two nodes can only connect using one edge of each color. In case the same type of connection should be established between two nodes, this framework can be combined with another one to reach this purpose [4].

Implementations of multilayered networks

In this section, we present to you the main types and implementations of the multilayer framework. For each implementation, we enumerate some of their applications in different fields.

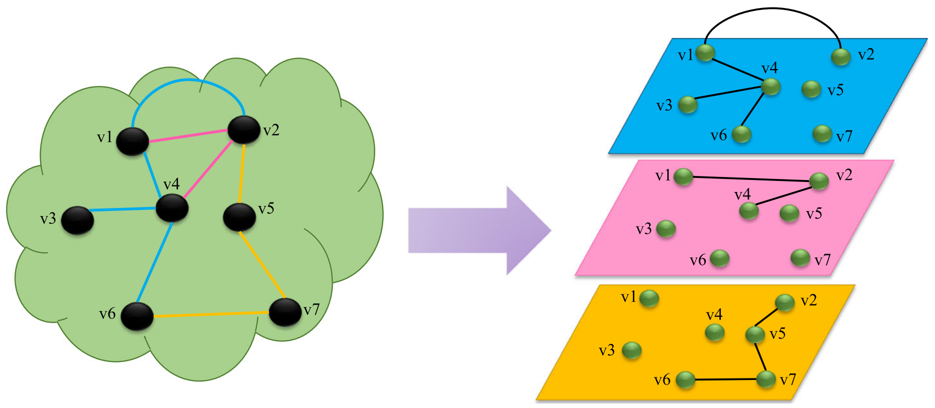


Fig. 2 Edge-colored Graphs. The nodes of the original networks are replicated over the different layers. The layers represent the different aspects of relationships in the network. Edges are divided among the layers based on their colors

Multiplex networks

Multiplex Networks (Fig. 3) are edge-colored graphs, where the nodes connect with edges belonging to M different aspects (Eq. 3). The edges are embedded based on their type in different layers all containing the complete set of nodes X of length n [4, 6–8]. Multiplex networks explore and incorporate the interconnectivity of the same system in multiple channels represented by the layers [9].

$$M layers, \qquad \{G_\alpha; \alpha \in \{1, \cdots, M\}\}, \qquad G_\alpha = (X_\alpha, E_\alpha), \ \ with X_\alpha = \{x_1, \cdots, x_N\} \tag{3}$$

A perfect example of a multiplex network is a social network, where different social interactions between individuals are studied, for example, friendship, vicinity, kinship, membership of the same cultural society, partnership or co-workership, etc. [8, 10]. The nodes, in this case, are the individuals, and the edges are the interactions between them. These networks were studied thoroughly over the past years since they are of great importance in affecting our daily lives. The study of these networks is particularly important for social media companies like Facebook, which helps them establish connections between their users and improve their services, including family connections, friendships, social interests, political views, etc., based on the information provided by the users, like backgrounds, demographics, etc. [11].

Another application presented by Cardillo et al. [12] introduces The European Air Transportation Multiplex Network, consisting of 37 layers representing the European airlines, N number of nodes representing the airports in Europe, and links between the nodes representing the direct flights.

Multilevel networks

Multilevel networks (Fig. 4) are edge-colored graphs. They are similar to multiplex networks, with a difference being that layers can contain not only a subset of edges but a subset of nodes as well. The graph (Eq. 4) is a tuple defined by the sets of nodes X, edges E, and layers S with length p.

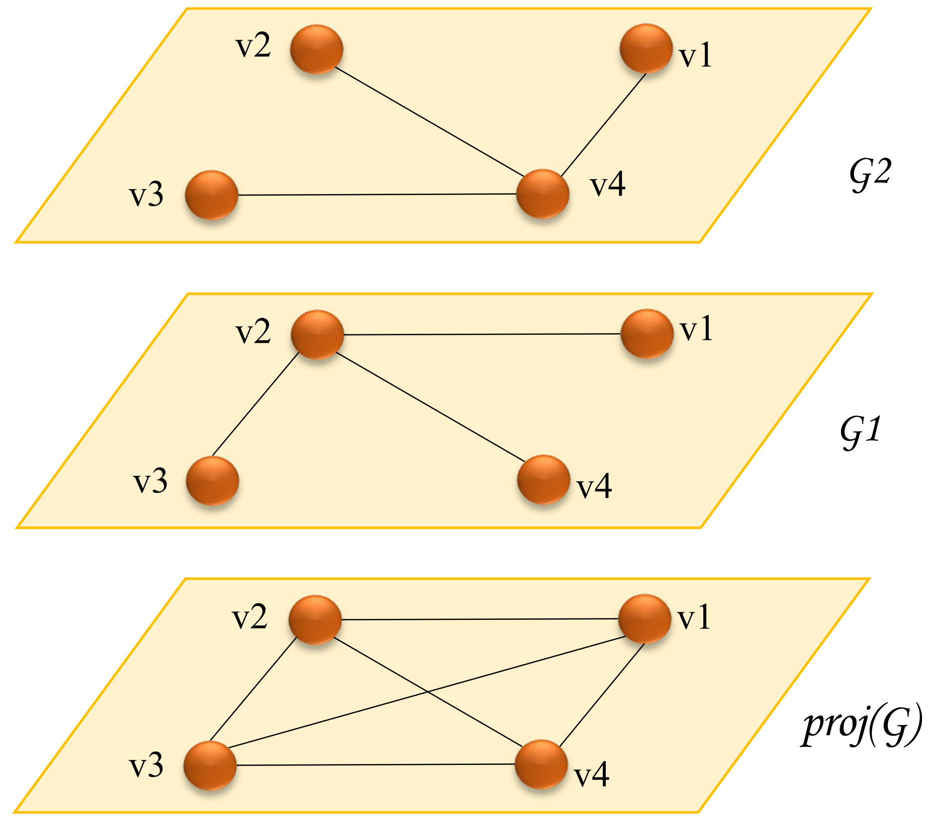


Fig. 3 A Multiplex Graph. The nodes of the network are replicated over layers, and each layer represents a particular aspect of connection between the nodes

$$\begin{gathered} M=(X,E,S) \\ S=\{S_1,\ldots,S_p\}\ \ sub\text{–}graphs \\ With\ \ S_j=(X_j,E_j), j=1,\ldots,p \\ X=\bigcup_{j=1}^{p} X_j \quad E=\bigcup_{j=1}^{p} E_j \end{gathered} \tag{4}$$

An example of the application of this framework is the Urban Transportation Network modeled by Aleta et al. [13]. In their paper, the authors present a multi-level model, representing various transportation methods in nine cities in Europe. The nodes are the different stops, while the layers are the different modes of transportation. This network is multilevel because the nodes or the stops do not exist on all the layers.

Multi-hypernetworks

Multi-Hypernetworks or hypergraphs (Fig. 5) are node-based graphs [1, 4], formed by different intersecting communities or sub-networks. A Multi-hypergraph (Eq. 5) is defined by a pair (X, H), X being the set of nodes, and H a multi-set of subsets of X being the edges [14]. These graphs focus on the nodes belonging to the same group rather than the connections between them. Nodes in the same subset are mapped to a layer with fully connected nodes. The notion

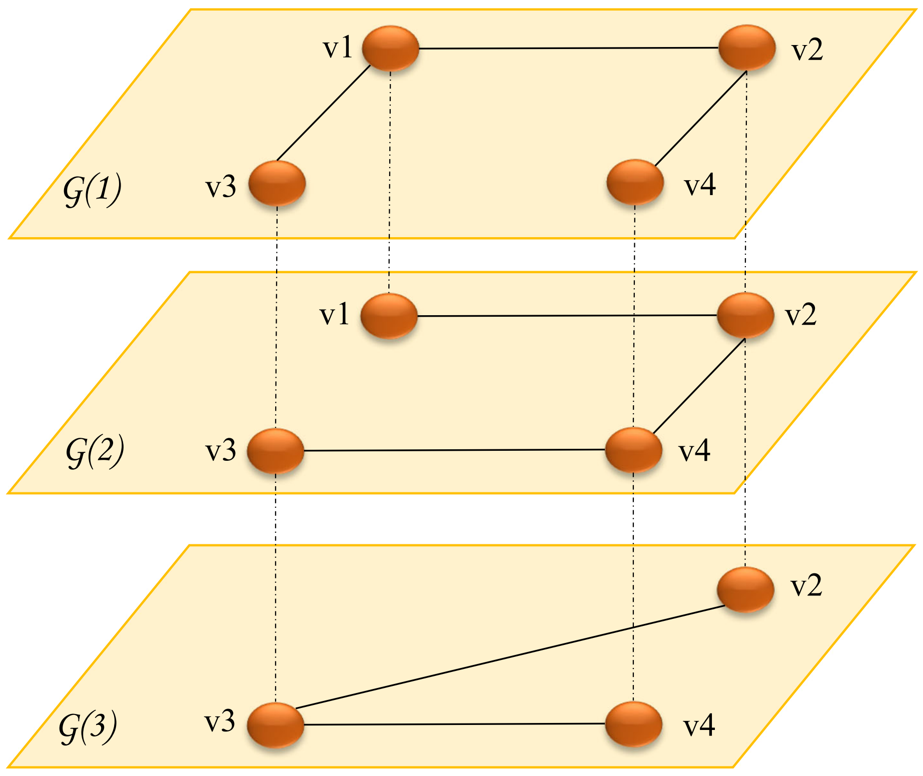


Fig. 4 Multilevel Graph. Each layer of the graph is a subset of the original network (nodes and intra-edges). The inter-edges only connect the same nodes on different layers

of edges here differs from normal networks; edges in these networks are called hyperedges and can connect multiple nodes at the same time [4, 15–17]. Each hyperedge is then mapped to a single layer. Multi-Hypernetworks are not layer-disjoint, but node-aligned since one node can be assigned to multiple layers, based on the intersection of the subsets.

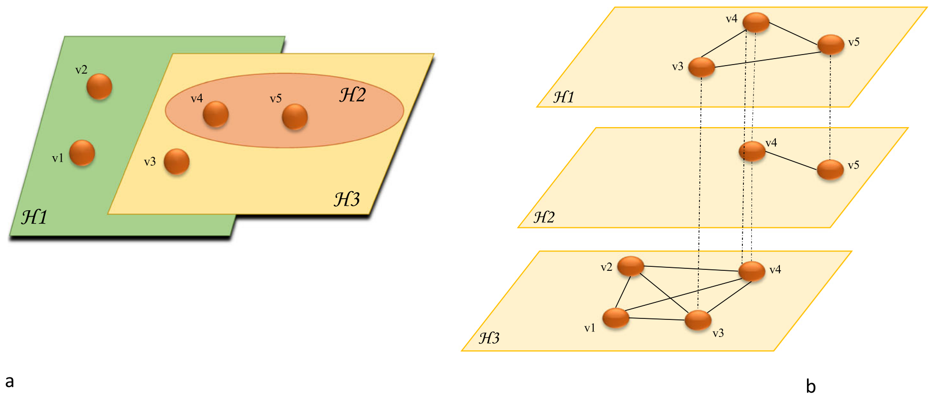


Fig. 5 Hypernetwork. Adapted from [1]. **a** The network in a monoplex view; **b** The network in a multilayer view. Nodes belonging to the same ensemble are assigned the same layer. All the nodes on one layer are intra-connected. The same nodes on different layers are interconnected (dotted edges)

$$\begin{aligned} \phi &= (X, H) \\ H &= \{H_1, \ldots H_p\} sub\text{–}sets \\ E_{\alpha\beta} &= \{(x, x); x \in X_\alpha \cap X_\beta\}, \text{where } \alpha, \beta \in \mathrm{H} \end{aligned} \tag{5}$$

Hypernetworks are optimal representations of networks with n-ary relationships since they allow hyperedges connecting more than 2 nodes at the same time [18]. They have been mainly used in Folksonomy, where a semantic structure is created by a collaborative annotation. An example of this application is Flickr, used to share multimedia between users, who are allowed to tag them. The network here can be viewed as a tripartite Hypernetwork consisting of three types of nodes being the users, the resources and the tags, and hyperedges connecting those three, meaning a user annotating a resource with a tag [1, 4, 16, 19].

Another application is presented by Chan and Hsu [20], where they discuss how the usage of hypernetworks in the service science can reveal hidden social structures in human networks, and therefore provide a better understanding of them.

Hypernetworks are also used in team sport networks. As presented by Ramos et al. [18], these networks can be used to study multiple aspects of interactions between the players, like spatiotemporal interactions, which inspects the dynamics of the game at different times and spots.

Interacting/interconnected/interdependent networks and networks of networks

This framework (Fig. 6) is a set of monoplex networks interacting with each other. Nodes can have intra-edges as well as inter-edges connecting them. The inter-edges E define the interactions between the L different monoplex networks G (Eq. 6), but a dependency in interdependent networks, i.e. some of the nodes in each network are dependent on nodes from others [1, 4, 21, 22]. The multilayer model of these networks is node-colored, layer-disjoint formed by embedding the different monoplex networks each into a single layer, then connecting them with their inter-edges. These networks help achieve global synchronization between different sub-systems [1].

$$\begin{aligned} &\{G_1, G_2, \ldots G_L\} \\ E_{\alpha\beta} &= interaction\ between\ G_\alpha\ and\ G_\beta \end{aligned} \tag{6}$$

Examples of such networks are transportation, telecommunication, electrical grids [23], and social networks [24]. The infrastructure network, in particular, is a perfectly fitting example, since it relies on electricity to function. This dependency between the networks makes this framework vulnerable since a failure in one part can cascade to affect the whole network. This problem has caught a lot of attention and many scientists are still studying it [22, 23, 25–28].

Multilayered networks and biology

Biological networks are difficult to explore because they contain a mass number of nodes connected in myriad ways. The best way to obtain information from these networks is to summarize the information about the nodes and links conveniently [29]. This summarization follows the criteria which we discussed in our first section but depends on the biological domain. Some domains require edge-colored methodologies to

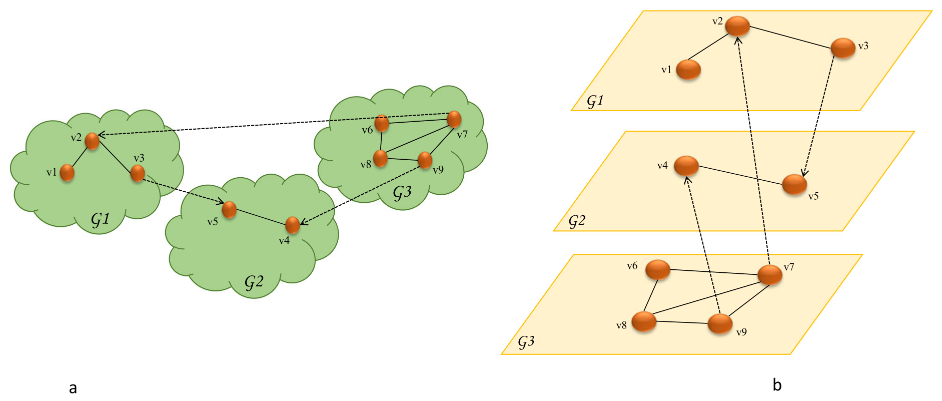


Fig. 6 Interacting/Interconnected Networks. Adapted from [1]. **a** The network in a monoplex view; **b** The network in a multilayer view. Nodes belonging to the same system are assigned the same layer. Inter- and intra-edges are kept the same

explore the different interactions between the molecules, like gene co-expression networks, or protein-protein interaction networks, or different behavior in cross-species networks. Others require node-colored frameworks, like cell networks to integrate interconnected and interdependent networks.

This usage is still in its infancy in some domains, like gene-expression networks, but very thriving in others, like epidemiology.

In this section, we summarize the usage of the multilayer framework in Biology with state-of-the-art work on this topic.

Protein-protein interaction networks

Protein-protein interaction (PPI) networks are collections of interactions between two or more proteins binding to carry a certain function [30], where nodes are the proteins and edges are physical or functional interactions between them [31]. These networks are heterogeneous in regard to the different types of interactions between them. The network could be seen as an edge-colored Multiplex or Multilevel network inspecting diverse aspects of interactions.

OhmNet [32] is an algorithm that transforms monoplex networks into a multilayer model. This algorithm aims at studying proteins in different tissues and learning their features. In this model, the tissues act as layers, and the proteins act as nodes. The model implemented by OhmNet was used in a recent publication by Kapadia et al. [33] to predict features of a multi-layer blood cell PPI network.

Multilayer PPI networks have also been used to study the life stages in species. In [34], the life stages in *Caenorhabditis elegans* are modeled in a node-colored multilayer network, where protein interactions are collected from various bioinformatics repositories, and then proteins occurring in different life cycles are distributed over six layers representing the different life stages.

Another Multilayer model introduced by Zhao et al. [35] aims at integrating PPI network, protein domain information, and protein complexes. The authors present a Multiplex edge-colored model consisting of three layers: the physical interaction layer (PIL), containing the protein-protein interactions, sharing domain layer (SDL), with proteins connected in case they share the same domain, and sharing complex layer (SCL), in case they are contained in the same complex. The nodes are the proteins replicated over the three layers (Fig. 7).

In addition, the High-Throughput data integration method implemented by Liang et al. [36] used an edge-colored Multiplex framework. The base of the graph is PPI networks from yeast and human interactomes which form the two layers, weighted with transcriptional data, and combined with biomedical knowledge.

Cell networks

Cell networks or pathways represent complex functions and reactions that occur in the cell. They describe interactions between genes, proteins, and metabolites [37]. To understand the cell structure, one needs to investigate not only these molecules and their intra-connectivity in isolation, but also their inter-connectivity. Gligorijević and Pržulj in [38] illustrate the cell network as an interdependent network of five layers: phenotypes, genome, proteome, metabolome, and transcriptome. The study of these different substrates results in a heterogeneous network formed of different types of nodes and interactions.

An example of a cell network is implemented by Liu et al. in [37], where they use a node-colored multilayer consisting of a gene-regulatory layer, PPI layer, and a metabolite layer. The genes are connected to their gene products on the PPI layer directly, and proteins are connected to metabolites in a many-to-many relation when they associate or participate in the same chemical reaction.

In [39], Rai et al. demonstrate how the usage of a multilayer framework helps to understand the behavior of cancer cells. In their paper, the authors study seven different cancer types (breast, oral, ovarian, cervical, lung, colon, and prostate), using a multilevel network model. Their model consists of seven layers for two networks, the normal and the disease networks, in which each layer reflects a cancer type. The nodes are the proteins in PPI networks, chosen based on their expression in normal and cancer cells. The common proteins are then extracted in three different sets: common in all normal cells, common in all cancer cells, and common in normal and disease cells (Fig. 8).

A similar work by Yu et al. [40] employs a multiplex network of three layers, for PPI networks in three different tissues: breast, prostate, and blood. They study the overlapping between the genes under the action of the drug trichostatin A (TSA) in three diseases it treated, leukemia, breast cancer, and prostate cancer. They identified two drug-target modules of TSA (M17, M18) as the potential treatment patterns of TSA.

Another paper by Li et al. [41] aims at identifying frequent coupled transcription-splicing modules. They build a two-layer interconnected model of gene co-expression networks and exon co-splicing networks. The layers contain respectively genes and exons, inter-connected if there is a relationship between them, and intra-connected if they are respectively co-expressed and co-spliced.

A tool also addressing data integration in Biology using a multilayer approach is Synergy Landscapes by Kuzmin et al. [42]. The tool constructs a multilayer model to connect researchers with resources through molecular interactions. The model integrates three networks: bio-medical co-authorship collaboration networks, molecules interacting in bioprocesses and papers describing them, and networks of bioprocesses involved in various diseases.

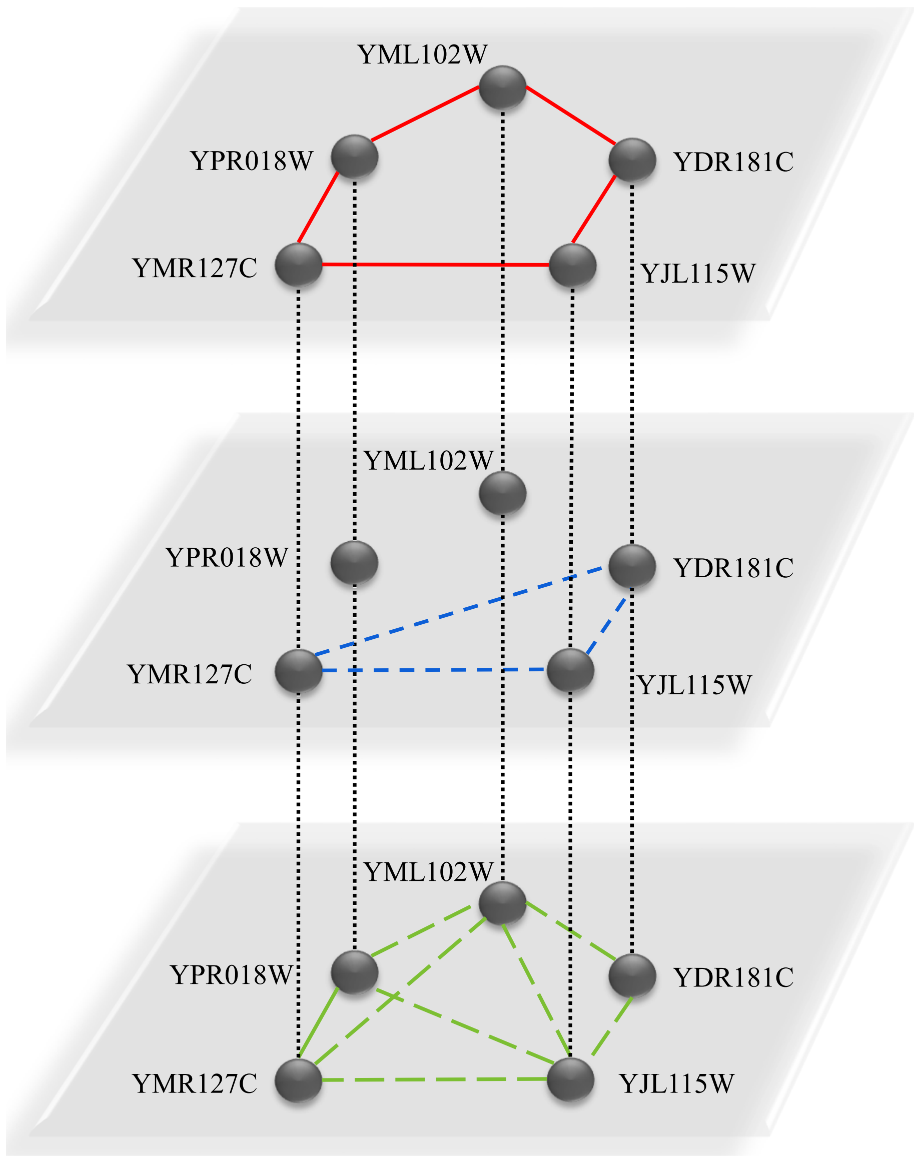


Fig. 7 Multilayer Protein Network. Adapted from [35]. The figure shows the edge-colored multiplex graph implemented by Zhao et al. [35], with three layers representing different interactions between the proteins

As an initiative for pathway data integration, we presented in a previous work a tool called mully [43]. The tool is an R package to create, modify, and visualize multilayer graphs. We aim at collecting pathway knowledge from different databases and embed the different elements into a node-colored multilayer model (Fig. 9).

One of the pathway databases is Reactome, which is manually curated, open-source, and fully downloadable [44]. Reactome contains BioPAX-encoded pathways that could be parsed using the R package rBiopaxParser implemented by Kramer et al. [45]. Another pathway resource is ndex, which is an online commons where scientists are able to upload and share networks [46]. The ndex servers can be queried using the R package ndexr implemented by Auer et al. [47], in order to download, modify, and upload networks.

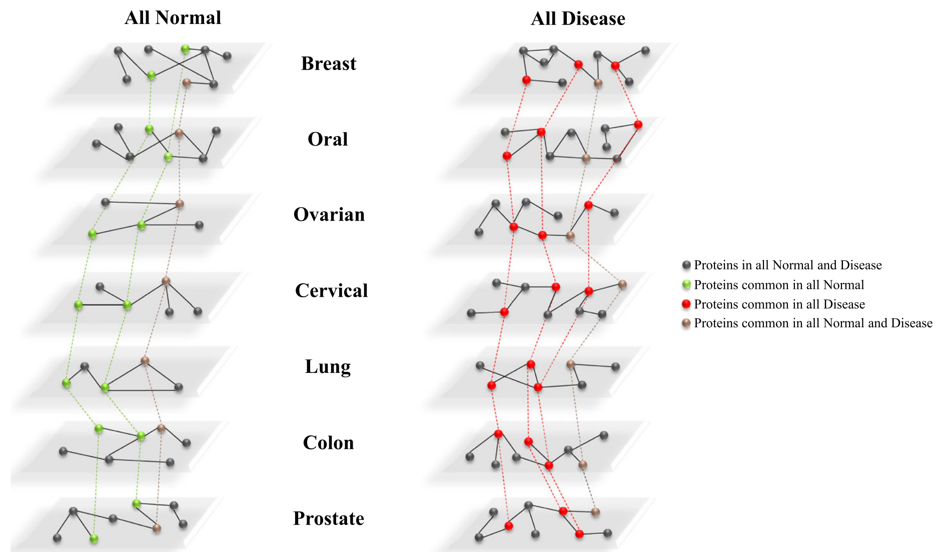

All Normal
All Disease
Breast
Oral
Ovarian
Cervical
Lung
Colon
Prostate
● Proteins in all Normal and Disease
● Proteins common in all Normal
● Proteins common in all Disease
● Proteins common in all Normal and Disease

Fig. 8 Multilayer Cancer Cells' Network. Adapted from [39]. The Figure shows two graphs, normal and disease. The nodes in these graphs are the proteins, assigned to seven layers reflecting seven cancer types

Gene expression networks

The complexity in genetic networks lies in the diversity of interactions between genes. A gene network is a collection of gene-gene regulatory connections [48]. It contains information on genes affecting each other with their expression levels. These networks can be inspected from a multiplex or a multilevel framework, with layers representing co-expression profiles in different settings including different time points, diseases, cells, or organisms, as well as from a Hypernetwork framework, with edges connecting more than two genes [48, 49].

In [50], Li et al. use a multiplex framework to illustrate co-expression networks to identify heavy recurrent subgraphs. The networks are extracted from microarray datasets, and each network is assigned to a layer.

In the paper describing the software muxViz [3], the authors also use genetic networks as an example to demonstrate their tool, used to visualize multiplex networks.

In another publication [51], Klosik et al. use an interdependent framework to study interactions between metabolites and genes. Their model consists of three layers: the gene regulatory layer, the protein-interface layer, and the metabolic layer.

Human neural networks

Neural networks are one of the most complex networks in the human body. A brain network is a structural network where nodes represent neural elements, i.e. the neurons or neuronal regions, and edges represent the physical connection between them, i.e. synapses or axonal projections. The connections in these networks are also affected by spatial factors, which means that the location and distance between the elements play a role in the probability of them being connected [52]. The modeling of brain networks on multiple levels is an evocative application, since they are continuously evolving, and different factors should be considered, such as temporal and spatial factors or different frequency bands. This model has been reviewed and adapted by many scientists to study the human brain in recent years [53–59].

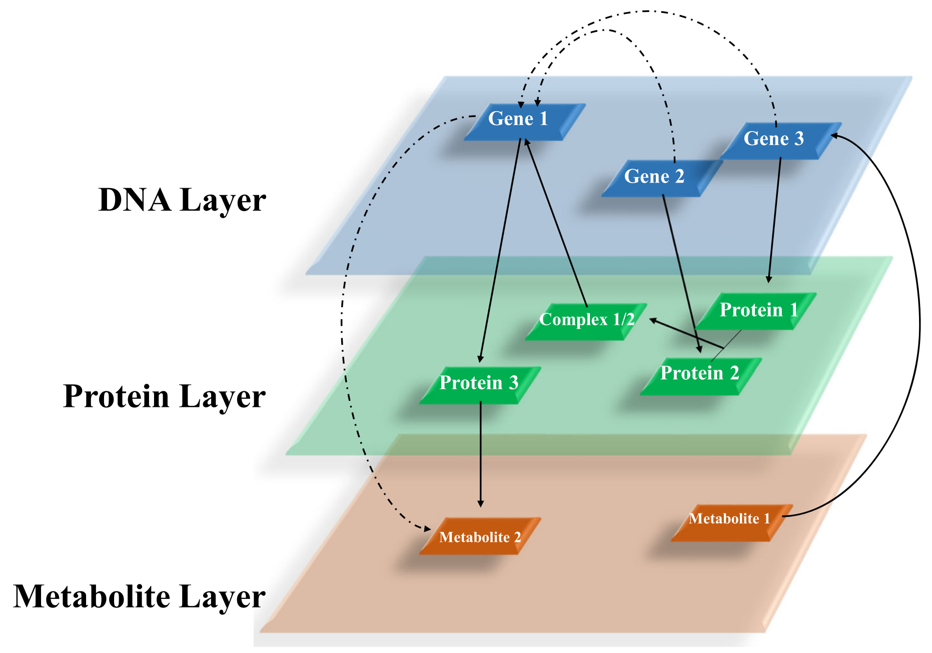


Fig. 9 A mully multilayer model. Each layer reflects a single type of the pathways' elements. The graph is node-colored

Jérémy Guillon presents in his Ph.D. thesis [60] a multiplex network of the human brain with Alzheimer's disease containing information from the brain at different frequency bands [61]. This model combines structural and functional connectivity in the brain and is used to detect core-periphery organizations [62].

Similarly, Dang et al. [63] present in their paper the Multivariate Time-Frequency Multilayer (MTFM) network, which is an edge-colored model to study the dynamics of two complex systems, namely oil-water flow, and driving fatigue in the brain. The model explores these dynamics by dividing the key frequency over L layers, each of which contains nodes representing the channel time-series, and edges representing mutual information connections. The driving fatigue in the brain MTFM network shows the changing of state in the brain during driving from alert to mental fatigue.

A similar model was also introduced by Brookes et al. [64] to study magnetoencephalography (MEG) data connectivity.

Spreading processes and disease behavior networks

Epidemiology is one of the biological fields in which the multilayer framework is used extensively. It is used to study the transmission and spreading of diseases between individuals, which is based on heterogeneous factors, like age, geographical locations, social status, and sexual behavior. These factors are used to assign the individuals of the population classes or types [65].

Examples of spreading networks are the epidemic models, SIR and SIS. In SIR, the individuals are assigned one of three traits: susceptible, meaning not yet infected but susceptible to infection, infected, meaning already carry the disease and can transmit it, and removed, meaning previously infected and do not transmit anymore. The infection starts with some individuals called seeds, and get transferred to adjacent or neighbor individuals at a certain time. So the nodes can go from susceptible to infected, or when cured to removed. In SIS, the individual only goes from infected to susceptible [65–69]. Although these models can be seen as node-colored graphs at different timestamps [70], they are mainly implemented as edge-colored. In [71] the authors use a two-layer interconnected model to study the interaction between two SIR-epidemics. Kivelä et al. [4] list a significant number of references on these implementations. An immunization strategy is also presented in [72] using the multiplex two-layer SIR model.

An adaptation of the SIR model is used by Riad et al. [73] to assess the risk of spreading of Ebola in Uganda. The authors propose a combination of SIR concept with a social network. Their model consists of two layers representing the interactions between the people, a permanent layer being family relations, and a temporal layer being the other general connections. The nodes, which are the individuals, can have either of two types (active and inactive) depending on if they carry the disease or not, and they are distributed in 23 districts.

In their publication [5], Allard et al. discuss some of the difficulties and problems faced in epidemiology when using simple models, like the iid hypothesis, and state the advantages of transforming them into multitype networks, using HIV as an example.

Mao and Yang [74] present another approach using multilayer networks, in which they use a two-layer multiplex model to demonstrate that infections and preventive behavior are transmitted simultaneously and affect each other. The individuals are replicated over both layers. The first layer contains the transmission relationships, and the second contains the interactions of influencing preventive behavior. Later the same author presents a similar model with an extra layer called Information, signifying whether the person was informed or uninformed of the disease [75] (Fig. 10). A similar approach is later used by Carnell et al. [76].

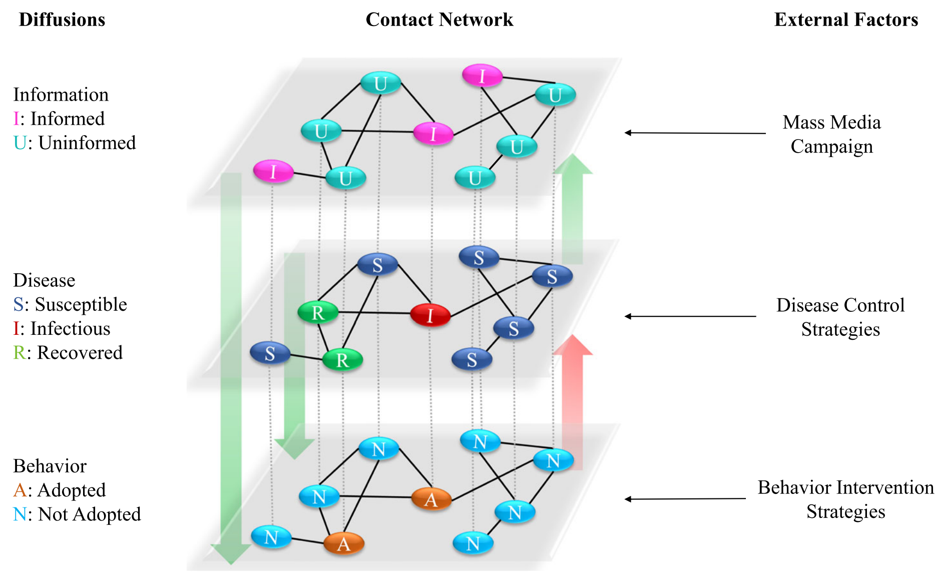


Fig. 10 A Multilayer Triple-diffusion Network. Adapted from [75]. The layers illustrate three diffusion processes. The nodes have different colors based on the state of the individuals in each network. The dotted lines connect the same nodes on the three layers. The arrows between the layers denote a positive (green) and negative (red) effect between the processes

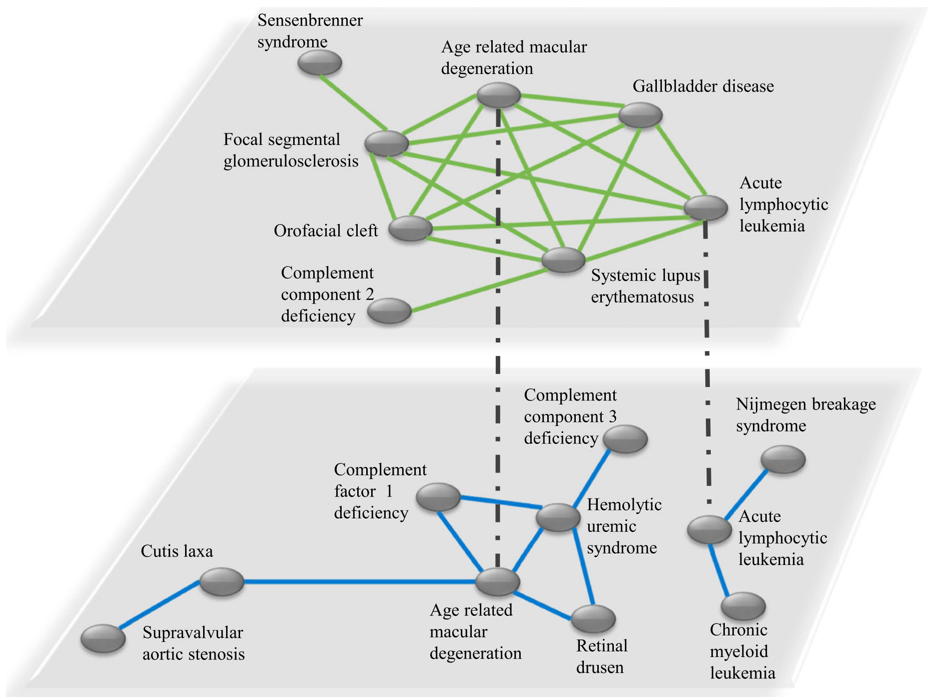

Sensenbrenner syndrome
Age related macular degeneration
Gallbladder disease
Focal segmental glomerulosclerosis
Acute lymphocytic leukemia
Orofacial cleft
Complement component 2 deficiency
Systemic lupus erythematosus
Complement component 3 deficiency
Nijmegen breakage syndrome
Complement factor 1 deficiency
Hemolytic uremic syndrome
Acute lymphocytic leukemia
Cutis laxa
Supravalvular aortic stenosis
Age related macular degeneration
Retinal drusen
Chronic myeloid leukemia

Fig. 11 Multiplex disease-disease interaction network. Adapted from [79]. The graph consists of two layers. The nodes are the diseases, connected on the first layer if they share a phenotypic interaction, and on the second if they share a genotypic one

dbNEI is a web-based knowledge resource that collects information related to the neuro-endocrine-immune systems (NEI) [77]. The database was later updated to dbNEI2.0, by adding new molecules and information. dbNEI2.0 builds a multilayer network for drug-NEI-disease. The model is node-colored, consisting of three layers, diseases, drugs, and NEI genes layers [78].

In other work, scientists were interested in disease-disease interactions and connections. Halu et al. present in [79] a two-layer model of diseases. On the genotype layer,

Table 1 Classification of the references based on the used implementation

Aspect	Implementation	Application Field	Reference
Node-colored	Interconnected	Cell Networks	[37, 41–43]
		Epidemiology	[78, 80]
	Interdependent	Cell Networks	[38]
		Gene Expression Networks	[51]
	Multitype	Epidemiology	[5, 34, 70]
Edge-Colored	Multiplex	PPI Networks	[35, 36]
		Cell Networks	[40]
		Gene Expression Networks	[3, 50]
		Human Neural Networks	[60–62]
		Epidemiology	[71–76, 79]
	Multilevel	PPI Networks	[32, 33]
		Cell Networks	[39]
		Human Neural Networks	[63, 64]

the diseases are linked in case they share a disease gene, while on the phenotype layer, they are linked if they share a symptom (Fig. 11).

Moreover, in [80], Yu et al. construct a four-layer similarity model of four disease interconnected networks: Human Disease Network Based on Protein Interaction Network (PIDN), Human Disease Similarity Network Based on Symptoms (DSDN), Gene Ontology- and Disease Ontology-Based Disease Similarity Networks (GODN and DODN). They aim at getting more accurate conserved disease modules and find potential disease-disease relationships.

In Table 1, the references are listed for each aspect, implementation and biomedical application of multilayered graphs.

Conclusions

The usage of Multilayer Graphs has proven to be very promising and prosperous in different fields of application, especially in Biology. Multilayer Graphs allow better inspection and representation of the topology and dynamics of heterogeneous systems compared to monoplex models [1]. In the case of biological networks, the heterogeneity lies in the diversity of interacting substrates, as well as in the various channels of interactions between them. Many scientists have adapted the multilayer model to solve complex biomedical problems in recent years, by using particular implementations for different situations. For instance, muxViz can be used to visualize edge-colored multilayer networks, such as multiplex or multilevel networks for gene expression data. Our tool mully can be used to create, modify, and visualize multi-omics data, like pathways, drugs, and diseases in a node-colored multilayer model [43].

The multilayer framework has many other applications yet to be discovered. Another approach is to mix different implementations in one model. For instance, a node-colored multilayer model built to integrate different omics data can contain five layers: protein, genes, metabolites, diseases, and drugs. However, it can be extended by dividing the protein and the gene layers to exploit gene co-expressions in multiple tissues or environment, and different protein-protein interactions from different experimental protocols [4]. This indeed helps reduce the loss of data and obtaining misleading results [3], detect patterns and correlations, and avoid hairball effects, resulting from the high number of interactions between the nodes.

Abbreviations

PPI: Protein-protein Interaction; TSA: Trichostatin A; Ph.D.: Philosophiae doctor; MTFM: Multivariate Time-Frequency Multilayer; MEG: Magnetoencephalography; SIR: Susceptible-Infected-Removed; SIS: Susceptible-Infected-Susceptible; HIV: Human immunodeficiency virus; NEI: Neuro-endocrine-immune

Acknowledgments

I would like to acknowledge the help of my friend Farah Diab, and of my colleagues Florian Auer, Dominik Müller, Peter Parys, Iñaki Soto Rey, Johann Frei, Samantha Cramer, Sophia Reichenzeller, and Alicia D'Angelo.

Authors' contributions

ZH contributed to Conceptualization, Investigation, and Writing – Original Draft Preparation. FK contributed to Funding Acquisition, Supervision, and Writing – Review & Editing. Both authors approved the submitted version of this paper.

Funding

This work is a part of the Multipath and coNfirm Projects funded by the GERMAN MINISTRY OF EDUCATION AND RESEARCH (Bundesministerium für Bildung und Forschung, BMBF) grant respectively FKZ01ZX1508 and FK01ZX1708D.

Availability of data and materials

Not Applicable.

Ethics approval and consent to participate
Not Applicable.

Consent for publication
Not Applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 30 January 2020 Accepted: 30 June 2020
Published online: 06 July 2020

References

1. Boccaletti S, Bianconi G, Criado R, del Genio CI, Gómez-Gardeñes J, Romance M, et al. The structure and dynamics of multilayer networks. Phys Rep. 2014;544(1):1–122. https://doi.org/10.1016/j.physrep.2014.07.001.
2. Traxl D, Boers N, Kurths J. Deep graphs - a general framework to represent and analyze heterogeneous complex systems across scales. Chaos. 2016;26(6):065303. https://doi.org/10.1063/1.4952963.
3. De Domenico M, Porter MA, Arenas A. MuxViz: a tool for multilayer analysis and visualization of networks. J Complex Netw. 2015;3(2):159–76. https://doi.org/10.1093/comnet/cnu038.
4. Kivelä M, Arenas A, Barthelemy M, Gleeson JP, Moreno Y, Porter MA. Multilayer networks. J Complex Netw. 2014;2(3): 203–71. https://doi.org/10.1093/comnet/cnu016.
5. Allard A, Noël P-A, Dubé LJ, Pourbohloul B. Heterogeneous bond percolation on multitype networks with an application to epidemic dynamics. Phys Rev E. 2009;79(3):036113. https://doi.org/10.1103/PhysRevE.79.036113.
6. Nicosia V, Bianconi G, Latora V, Barthelemy M. Growing multiplex networks. Phys Rev Lett. 2013;111(5):058701. https://doi.org/10.1103/PhysRevLett.111.058701.
7. Battiston F, Nicosia V, Latora V. Structural measures for multiplex networks. Phys Rev E. 2014;89(3):032804. https://doi.org/10.1103/PhysRevE.89.032804.
8. Bianconi G. Statistical mechanics of multiplex networks: entropy and overlap. Phys Rev E. 2013;87(6):062806. https://doi.org/10.1103/PhysRevE.87.062806.
9. De Domenico M, Solè-Ribalta A, Cozzo E, Kivelä M, Moreno Y, Porter MA, et al. Mathematical formulation of multi-layer networks. Phys Rev X. 2013;3(4). https://doi.org/10.1103/PhysRevX.3.041022.
10. Solá L, Romance M, Criado R, Flores J, García del Amo A, Boccaletti S. Eigenvector centrality of nodes in multiplex networks. Chaos. 2013;23(3):033131. https://doi.org/10.1063/1.4818544.
11. Lewis K, Kaufman J, Gonzalez M, Wimmer A, Christakis N. Tastes, ties, and time: a new social network dataset using Facebook.com. Soc Networks. 2008;30(4):330–42. https://doi.org/10.1016/j.socnet.2008.07.002.
12. Cardillo A, Gómez-Gardeñes J, Zanin M, Romance M, Papo D, del Pozo F, et al. Emergence of network features from multiplexity. Sci Rep. 2013;3:1344. https://doi.org/10.1038/srep01344.
13. Aleta A, Meloni S, Moreno Y. A Multilayer perspective for the analysis of urban transportation systems. Sci Rep. 2017;7(1): 1–9. https://doi.org/10.1038/srep44359.
14. Pearson KJ, Zhang T. On spectral hypergraph theory of the adjacency tensor. Graphs Combinations. 2014;30(5):1233–48. https://doi.org/10.1007/s00373-013-1340-x.
15. Zlatić V, Ghoshal G, Caldarelli G. Hypergraph topological quantities for tagged social networks. Phys Rev E Stat Nonlinear Soft Matter Phys. 2009;80(3 Pt 2):036118. https://doi.org/10.1103/PhysRevE.80.036118.
16. Cromar GL, Zhao A, Yang A, Parkinson J. Hyperscape: visualization for complex biological networks. Bioinformatics. 2015; 31(20):3390–1. https://doi.org/10.1093/bioinformatics/btv385.
17. Thai MT, Wu W, Xiong H. Big Data in Complex and Social Networks (1st. ed.). Chapman & Hall/CRC. 2016;ISBN:978-1-4987-2684-9.
18. Ramos J, Lopes RJ, Marques P, Araújo D. Hypernetworks reveal compound variables that capture cooperative and competitive interactions in a soccer match. Front Psychol. 2017;8:1379. https://doi.org/10.3389/fpsyg.2017.01379.
19. Ghoshal G, Zlatić V, Caldarelli G, Newman MEJ. Random hypergraphs and their applications. Phys Rev E. 2009;79(6): 066118. https://doi.org/10.1103/PhysRevE.79.066118.
20. Chan WKV, Hsu C. How hyper-network analysis helps understand human networks? Serv Sci. 2010;2(4):270–80. https://doi.org/10.1287/serv.2.4.270.
21. Donges JF, Schultz HCH, Marwan N, Zou Y, Kurths J. Investigating the topology of interacting networks: theory and application to coupled climate subnetworks. Eur Phys J B. 2011;84(4):635–51. https://doi.org/10.1140/epjb/e2011-10795-8.
22. Baxter GJ, Dorogovtsev SN, Goltsev AV, Mendes JFF. Avalanche collapse of interdependent networks. Phys Rev Lett. 2012;109(24):248701. https://doi.org/10.1103/PhysRevLett.109.248701.
23. Almoghathawi Y, Barker K, Albert LA. Resilience-driven restoration model for interdependent infrastructure networks. Reliab Eng Syst Safe. 2019;185:12–23. https://doi.org/10.1016/j.ress.2018.12.006.
24. Wang B, Chen X, Wang L. Probabilistic interconnection between interdependent networks promotes cooperation in the public goods game. J Stat Mech. 2012;2012(11):P11017. https://doi.org/10.1088/1742-5468/2012/11/P11017.
25. Buldyrev SV, Parshani R, Paul G, Stanley HE, Havlin S. Catastrophic cascade of failures in interdependent networks. Nature. 2010;464(7291):1025–8. https://doi.org/10.1038/nature08932.
26. Jin Q, Wang L, Xia C-Y, Wang Z. Spontaneous symmetry breaking in interdependent networked game. Sci Rep. 2014;4: 4095. https://doi.org/10.1038/srep04095.
27. Wang Z, Szolnoki A, Perc M. Optimal interdependence between networks for the evolution of cooperation. Sci Rep. 2013;3:2470. https://doi.org/10.1038/srep02470.
28. Vespignani A. The fragility of interdependency. Nature. 2010;464(7291):984–5. https://doi.org/10.1038/464984a.
29. Guimerà R, Nunes Amaral LA. Functional cartography of complex metabolic networks. Nature. 2005;433(7028):895–900. https://doi.org/10.1038/nature03288.

30. Andreopoulos W, Labudde D. Protein-protein interaction networks. 2013. http://www.bioforscher.de/bigM/ippb9076rp8sityx/manager/documents/general/pdf/books/chapters/protein_protein_interaction_networks.pdf. Accessed 2 Dec 2019.
31. Safari-Alighiarloo N, Taghizadeh M, Rezaei-Tavirani M, Goliaei B, Peyvandi AA. Protein-protein interaction networks (PPI) and complex diseases. Gastroenterol Hepatol Bed Bench. 2014;7(1):17–31.
32. Zitnik M, Leskovec J. Predicting multicellular function through multi-layer tissue networks. Bioinformatics. 2017;33(14): i190–8. https://doi.org/10.1093/bioinformatics/btx252.
33. Kapadia P, Khare S, Priyadarshini P, Das B. Predicting protein-protein interaction in multi-layer blood cell PPI networks. In: Luhach AK, Jat DS, Hawari KBG, Gao X-Z, Lingras P, editors. Advanced informatics for computing research. ICAICR 2019. Communications in computer and information science, vol. 1076. Singapore: Springer; 2019. p. 240–51. https://doi.org/10.1007/978-981-15-0111-1_22.
34. Shinde P, Jalan S. A multilayer protein-protein interaction network analysis of different life stages in Caenorhabditis elegans. EPL. 2015;112(5):58001. https://doi.org/10.1209/0295-5075/112/58001.
35. Zhao B, Hu S, Li X, Zhang F, Tian Q, Ni W. An efficient method for protein function annotation based on multilayer protein networks. Hum Genomics. 2016;10(1):33. https://doi.org/10.1186/s40246-016-0087-x.
36. Liang L, Chen V, Zhu K, Fan X, Lu X, Lu S. Integrating data and knowledge to identify functional modules of genes: a multilayer approach. BMC Bioinformatics. 2019;20(225). https://doi.org/10.1186/s12859-019-2800-y.
37. Liu X, Maiorino E, Halu A, Loscalzo J, Gao J, Sharma A. Robustness and lethality in multilayer biological molecular networks. bioRxiv. 2019;818963. https://doi.org/10.1101/818963.
38. Gligorijević V, Pržulj N. Methods for biological data integration: perspectives and challenges. J R Soc Interface. 2015; 12(112). https://doi.org/10.1098/rsif.2015.0571.
39. Rai A, Pradhan P, Nagraj J, Lohitesh K, Chowdhury R, Jalan S. Understanding cancer complexome using networks, spectral graph theory and multilayer framework. Sci Rep. 2017;7(1):1–16. https://doi.org/10.1038/srep41676.
40. Yu L, Shi Y, Zou Q, Gao L. Studying the drug treatment pattern based on the action of drug and multi-layer network model. bioRxiv. 2019;780858. https://doi.org/10.1101/780858.
41. Li W, Dai C, Liu C-C, Zhou XJ. Algorithm to identify frequent coupled modules from two-layered network series: application to study transcription and splicing coupling. J Comput Biol. 2012;19(6):710–30. https://doi.org/10.1089/cmb.2012.0025.
42. Kuzmin K, Gaiteri C, Szymanski BK. Synergy landscapes: a multilayer network for collaboration in biological research. In: Wierzbicki A, Brandes U, Schweitzer F, Pedreschi D, editors. Advances in network science. NetSci-X 2016. Lecture notes in computer science, vol. 9564. Cham: Springer; 2016. p. 205–12. https://doi.org/10.1007/978-3-319-28361-6_18.
43. Hammoud Z, Kramer F. mully: an R package to create, modify and visualize multilayered graphs. Genes (Basel). 2018; 9(11):519. https://doi.org/10.3390/genes9110519.
44. Croft D, Mundo AF, Haw R, Milacic M, Weiser J, Wu G, et al. The Reactome pathway knowledgebase. Nucleic Acids Res. 2014;42(D1):D472–7. https://doi.org/10.1093/nar/gkt1102.
45. Kramer F, Bayerlová M, Klemm F, Bleckmann A, Beissbarth T. rBiopaxParser--an R package to parse, modify and visualize BioPAX data. Bioinformatics. 2013;29(4):520–2. https://doi.org/10.1093/bioinformatics/bts710.
46. Pratt D, Chen J, Welker D, Rivas R, Pillich R, Rynkov V, et al. NDEx, the network data exchange. Cell Syst. 2015;1(4):302–5. https://doi.org/10.1016/j.cels.2015.10.001.
47. Auer F, Hammoud Z, Ishkin A, Pratt D, Ideker T, Kramer F. ndexr-an R package to interface with the network data exchange. Bioinformatics. 2018;34(4):716–7. https://doi.org/10.1093/bioinformatics/btx683.
48. Brazhnik P, de la Fuente A, Mendes P. Gene networks: how to put the function in genomics. Trends Biotechnol. 2002; 20(11):467–72. https://doi.org/10.1016/S0167-7799(02)02053-X.
49. Penfold CA, Wild DL. How to infer gene networks from expression profiles, revisited. Interface Focus. 2011;1(6):857–70. https://doi.org/10.1098/rsfs.2011.0053.
50. Li W, Liu C-C, Zhang T, Li H, Waterman MS, Zhou XJ. Integrative analysis of many weighted co-expression networks using tensor computation. PLoS Comput Biol. 2011;7(6):e1001106. https://doi.org/10.1371/journal.pcbi.1001106.
51. Klosik DF, Grimbs A, Bornholdt S, Hütt M-T. The interdependent network of gene regulation and metabolism is robust where it needs to be. Nat Commun. 2017;8(1):1–9. https://doi.org/10.1038/s41467-017-00587-4.
52. Bullmore E, Sporns O. Complex brain networks: graph theoretical analysis of structural and functional systems. Nat Rev Neurosci. 2009;10(3):186–98. https://doi.org/10.1038/nrn2575.
53. Muldoon SF, Bassett DS. Network and multilayer network approaches to understanding human brain dynamics. Philos Sci. 2016;83(5):710–20. https://doi.org/10.1086/687857.
54. Mandke K, Meier J, Brookes MJ, O'Dea RD, Van Mieghem P, Stam CJ, et al. Comparing multilayer brain networks between groups: introducing graph metrics and recommendations. NeuroImage. 2018;166:371–84. https://doi.org/10.1016/j.neuroimage.2017.11.016.
55. Puxeddu MG, Petti M, Mattia D, Astolfi L. The optimal setting for multilayer modularity optimization in multilayer brain networks*. In: 2019 41st annual international conference of the IEEE engineering in medicine and biology society (EMBC), Berlin, Germany; 2019. p. 624–7. https://doi.org/10.1109/EMBC.2019.8856674.
56. Pedersen M, Zalesky A, Omidvarnia A, Jackson GD. Reply to Yang et al.: multilayer network switching and behavior. PNAS. 2019;116(34):16673. https://doi.org/10.1073/pnas.1910493116.
57. Pedersen M, Zalesky A, Omidvarnia A, Jackson GD. Brain connectivity dynamics: multilayer network switching rate predicts brain performance. bioRxiv. 2018;403105. https://doi.org/10.1101/403105.
58. Vaiana M, Muldoon S. Multilayer Brain Networks. J Nonlinear Sci. 2018. https://doi.org/10.1007/s00332-017-9436-8.
59. De Domenico M. Multilayer modeling and analysis of human brain networks. Gigascience. 2017;6(5):1–8. https://doi.org/10.1093/gigascience/gix004.
60. Guillon J. Multilayer approach to brain connectivity in Alzheimer's disease. Neuroscience. Pierre and Marie Curie University, 2018. English. tel-01985286. https://tel.archives-ouvertes.fr/tel-01985286/file/PhD_Thesis_v2.0.pdf.
61. Guillon J, Attal Y, Colliot O, La Corte V, Dubois B, Schwartz D, et al. Loss of brain inter-frequency hubs in Alzheimer's disease. Sci Rep. 2017;7(1):10879. https://doi.org/10.1038/s41598-017-07846-w.
62. Battiston F, Guillon J, Chavez M, Latora V, De Vico Fallani F. Multiplex core-periphery organization of the human connectome. J R Soc Interface. 2018;15(146):20180514. https://doi.org/10.1098/rsif.2018.0514.
63. Dang W, Gao Z, Lv D, Liu M, Cai Q, Hong X. A novel time-frequency multilayer network for multivariate time series analysis. New J Phys. 2018;20(12):125005. https://doi.org/10.1088/1367-2630/aaf51c.

64. Brookes MJ, Tewarie PK, Hunt BAE, Robson SE, Gascoyne LE, Liddle EB, et al. A multi-layer network approach to MEG connectivity analysis. NeuroImage. 2016;132:425–38. https://doi.org/10.1016/j.neuroimage.2016.02.045.
65. Vazquez A. Spreading of infectious diseases on heterogeneous populations: multi-type network approach. Phys Rev E. 2006;74(6):066114. https://doi.org/10.1103/PhysRevE.74.066114.
66. Salehi M, Sharma R, Marzolla M, Magnani M, Siyari P, Montesi D. Spreading processes in multilayer networks. IEEE Trans Netw Sci Eng. 2015;2(2):65–83. https://doi.org/10.1109/TNSE.2015.242596.
67. Saumell-Mendiola A, Serrano MÁ, Boguñá M. Epidemic spreading on interconnected networks. Phys Rev E. 2012;86(2): 026106. https://doi.org/10.1103/PhysRevE.86.026106.
68. Sahneh FD, Scoglio C, Chowdhury FN. Effect of coupling on the epidemic threshold in interconnected complex networks: a spectral analysis. In: 2013 American Control Conference, Washington, DC; 2013. p. 2307–12. https://doi.org/10.1109/ACC.2013.6580178.
69. de Arruda GF, Cozzo E, Peixoto TP, Rodrigues FA, Moreno Y. Disease localization in multilayer networks. Phys Rev X. 2017;7(1):011014. https://doi.org/10.1103/PhysRevX.7.011014.
70. Hindes J, Singh S, Myers CR, Schneider DJ. Epidemic fronts in complex networks with metapopulation structure. Phys Rev E. 2013;88(1):012809. https://doi.org/10.1103/PhysRevE.88.012809.
71. Zhou S, Xu S, Wang L, Liu Z, Chen G, Wang X. Propagation of interacting diseases on multilayer networks. Phys Rev E. 2018;98(1–1):012303. https://doi.org/10.1103/PhysRevE.98.012303.
72. Singh V, Verma P, Muthukumaar V, Kumar V, Tewari M, Lai K-K, et al. Immunization strategy for epidemic spreading based on membership (m) over a multilayer network. Bus Strategy Dev. 2019. https://doi.org/10.1002/bsd2.87.
73. Riad MH, Sekamatte M, Ocom F, Makumbi I, Scoglio CM. Risk assessment of Ebola virus disease spreading in Uganda using a multilayer temporal network. bioRxiv. 2019;645598. https://doi.org/10.1101/645598.
74. Mao L, Yang Y. Coupling infectious diseases, human preventive behavior, and networks – a conceptual framework for epidemic modeling. Soc Sci Med. 2012;74(2):167–75. https://doi.org/10.1016/j.socscimed.2011.10.012.
75. Mao L. Modeling triple-diffusions of infectious diseases, information, and preventive behaviors through a metropolitan social network—an agent-based simulation. Appl Geogr. 2014;50:31–9. https://doi.org/10.1016/j.apgeog.2014.02.005.
76. Granell C, Gómez S, Arenas A. Dynamical interplay between awareness and epidemic spreading in multiplex networks. Phys Rev Lett. 2013;111(12):128701. https://doi.org/10.1103/physrevlett.111.128701.
77. Zhuang Y, Li S, Li Y. dbNEI: a specific database for neuro-endocrine-immune interactions. Neuro Endocrinol Lett. 2006; 27(1–2):53–9.
78. Zhang J, Ma T, Li Y, Li S. dbNEI2.0: building multilayer network for drug–NEI–disease. Bioinformatics. 2008;24(20):2409–11. https://doi.org/10.1093/bioinformatics/btn388.
79. Halu A, De Domenico M, Arenas A, Sharma A. The multiplex network of human diseases. NPJ Syst Biol Appl. 2019;5(1):1–12. https://doi.org/10.1038/s41540-019-0092-5.
80. Yu L, Yao S, Gao L, Zha Y. Conserved disease modules extracted from multilayer heterogeneous disease and gene networks for understanding disease mechanisms and predicting disease treatments. Front Genet. 2019. https://doi.org/10.3389/fgene.2018.00745.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Ready to submit your research? Choose BMC and benefit from:

- fast, convenient online submission
- thorough peer review by experienced researchers in your field
- rapid publication on acceptance
- support for research data, including large and complex data types
- gold Open Access which fosters wider collaboration and increased citations
- maximum visibility for your research: over 100M website views per year

At BMC, research is always in progress.

Learn more biomedcentral.com/submissions

BMC

Appendix B

Publication “mully: an R Package to create, modify and visualize multilayered networks”

genes

MDPI

Article

mully: An R Package to Create, Modify and Visualize Multilayered Graphs

Zaynab Hammoud [1,2,*] and Frank Kramer [2]

1 Department of Medical Statistics, University Medical Center Göttingen, Humboldtallee 32, 37073 Göttingen, Germany
2 Institute of Computer Science, IT Infrastructure for Translational Medical Research, University of Augsburg, Universitätsstraße 6a, 86159, Augsburg, Germany; frank.kramer@informatik.uni-augsburg.de
* Correspondence: zaynab.hammoud@med.uni-goettingen.de

Received: 1 October 2018; Accepted: 18 October 2018; Published: 23 October 2018

Abstract: The modelling of complex biological networks such as pathways has been a necessity for scientists over the last decades. The study of these networks also imposes a need to investigate different aspects of nodes or edges within the networks, or other biomedical knowledge related to it. Our aim is to provide a generic modelling framework to integrate multiple pathway types and further knowledge sources influencing these networks. This framework is defined by a multi-layered model allowing automatic network transformations and documentation. By providing a tool that generates this model, we aim to facilitate the data integration, boost the reproducibility and increase the interoperability between different sources and databases in the field of pathways. We present *mully* R package that allows the user to create, modify and visualize graphs with multi-layers. The package is implemented with features to specifically handle multilayered graphs.

Keywords: multilayered graphs; modelling in systems medicine; pathway modelling; pathway data integration; network visualization

1. Introduction

Network theory has been used for many years in the modelling and analysis of complex systems, as epidemiology, biology and biomedicine [1,2]. As the data evolves and become more heterogeneous and complex, monoplex networks become an oversimplification of the corresponding systems [1]. This imposes a need to go beyond traditional networks into a richer framework capable of hosting objects and relations of different scales and attributes [3,4], called Multilayered Network. These complex networks have contributed in many contexts and fields [5], although they have been rarely exploited in the investigation of biological networks, where their application seems very convenient [2].

In order to fill this gap, we present a multilayer framework that can be applicable in various domains, especially in the field of network modelling.

Our idea is to integrate pathways and their related knowledge into a multilayer model, where each layer represents one of their elements. The model offers a feature we call "Selective Inclusion of Knowledge", as well as a collection of related knowledge into a single graph, like diseases and drugs.

The final aim is to provide a reproducible approach to integrate prior biomedical knowledge about networks in personalized medicine algorithms [1].

In this paper, we present an R software that we call *mully* (**mul**ti**l**a**y**ered graphs) that serves our objective by generating a generic model used for data integration. *mully* implements the multi-layer models within R and will subsequently be extended to parse various databases and further knowledge sources.

This paper consists of 3 main sections: in the first section, we give an overview on multilayered graphs and their implementation, a description of the model implemented in this package, as well as brief explanation about the implementation process of the package. The second is the Result section, where we highlight the most important features offered by the *mully* Package. Finally, in the third section we conclude.

2. Materials and Methods

2.1. Multilayered Graphs

Multilayered graphs are the new trend in Graph Theory used by a large number of scientists nowadays. They are employed in the modelling of big networks, with heterogeneous nodes (vertices) or relations (edges). Considering that this framework has many applications and in different fields, its interpretation and implementation depend on the subject that it's serving. The main difference between these types is the criteria to link a node to a layer. In this context, two main types of networks can be distinguished: Node-colored graphs (NCGs) and edge-coloured graphs (ECGs) [1]. The following figures (Figure 1a,b) explain how to derive layers from regular graphs. On the left (Figure 1a) is an ECG, which is a graph with heterogeneous relations between the vertices. To transform this type of graphs into multilayered graphs, the nodes are replicated over all the layers, and each layer contains a subset of relative edges.

On the other hand, NCGs (Figure 1b) are graphs where nodes have aspects or types defined by colours. In order to build the multilayered graph from a NCG, the nodes are mapped to layers, leading to a network of networks, i.e., nodes having the same colours are grouped in the same layer. These graphs are usually layered-disjoint, i.e., the nodes can only be mapped to a single layer.

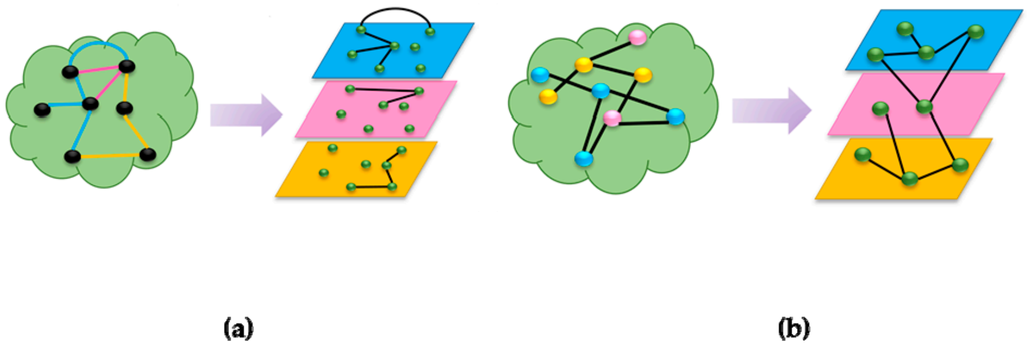


Figure 1. Two categories of multilayered graphs. (**a**) Edge-colored Graphs (ECG) (**b**) Node-colored Graphs (NCG).

The general model implemented in *mully* is a layered-disjoint NCG, and can be either directed or undirected. The following figure (Figure 2) gives an example of the generic model implemented in our package, which constitutes of n layers of nodes, connected with inter- and intra-layer edges.

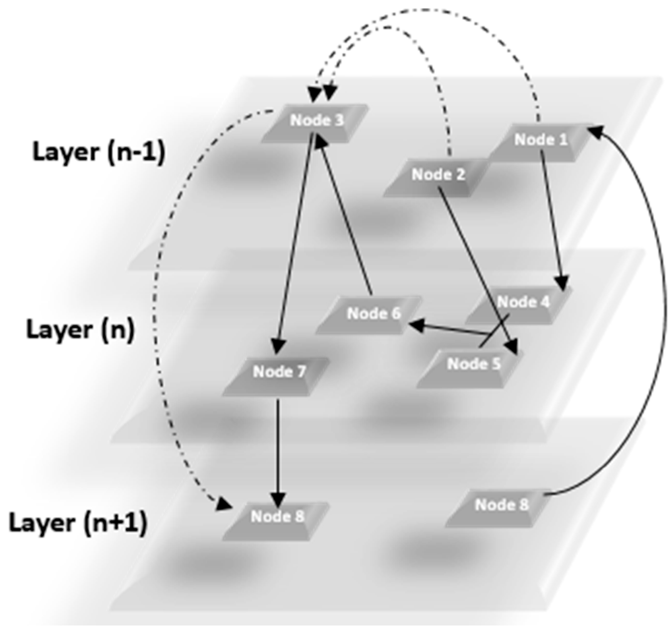


Figure 2. Generic Model of the Multilayered Graph implemented in our package.

2.2. Dependencies

The *mully* package is an R package that inherits the igraph object from the igraph R package [6] and adds additional information concerning the layers and other needed attributes. This package consists of a set of functions for nodes, edges, layers, graphs and visualization. The package is also set with a demo function that creates a sample graph used in order to try it.

It uses functions from other packages, for instance the 3D visualization is generated using the rgl R package [7], with some modifications applied concerning the layouting. It also refers to the RCX/ndexr package [8] to export the *mully* graph in Cytoscape Cyberinfrastructure Network Interchange Format (CX) [9] using its constructor.

3. Result and Features

The *mully* R package provides all the functionalities to work with graphs, which we call Standard Operations (Figure 3). In addition, we implemented special features to ease this work and the handling of big data import.

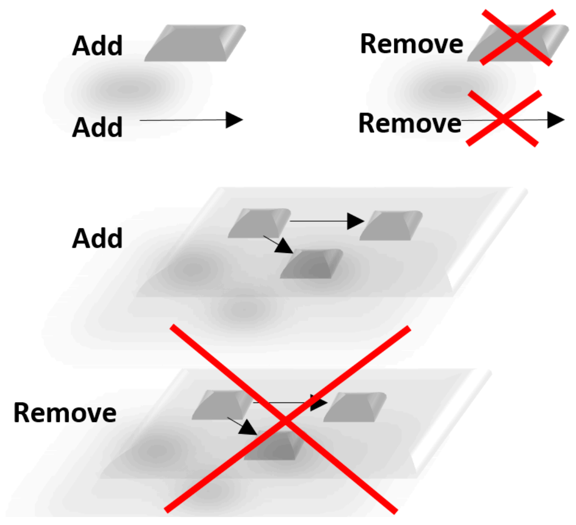


Figure 3. Standard Operations in *mully*.

The main features are: transitivity, smart merging, undoing, visualization and converters.

3.1. Transitivity

One of the important functionalities offered by *mully* is the transitivity. When choosing to delete one or a set of nodes (a whole layer for example), the user can select to add the transitive edge before removing the incident ones. This feature is required in order to preserve the routes, especially when working with structured networks. In the following figure (Figure 4), we show an example of the impact of removing a node after choosing the transitive option.

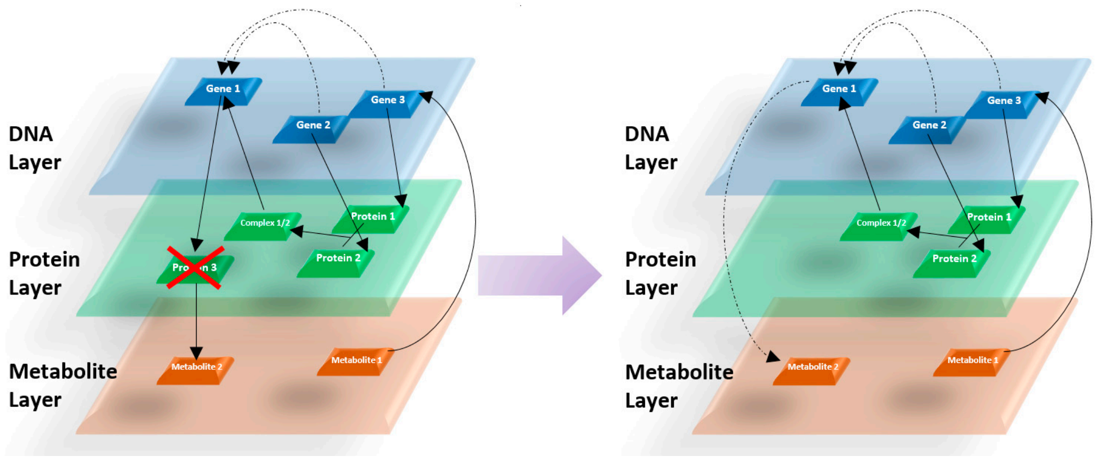


Figure 4. Addition of transitive edges after the deletion of a certain node.

3.2. Merging

In order to provide an easy use of our package, we provide a smart merging function to create a single valid *mully* graph out of two inputs. The merge is based on the layers, i.e., the nodes from both input graphs are combined based on their assigned layers. This merging prevents the replication of data, for example in multiple sources, by monitoring the nodes and edges with same attributes (name, labels, etc.). Figure 5 shows a 2D visualization of two *mully* graphs, and the result of their merging.

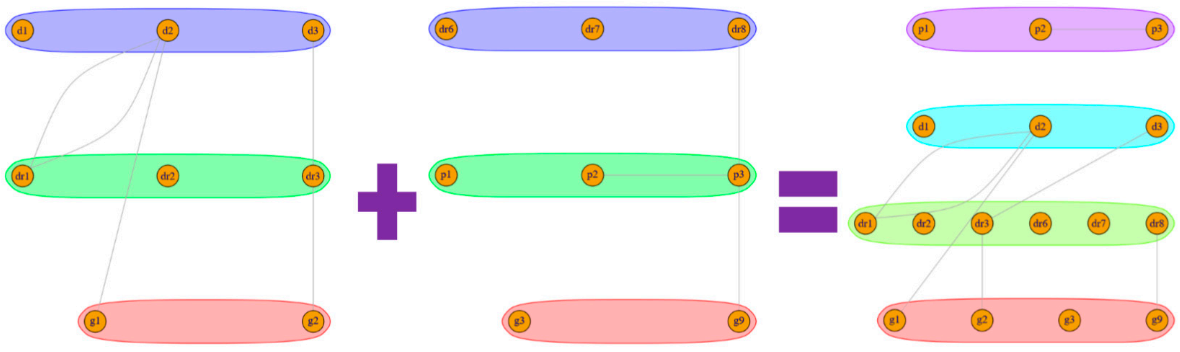


Figure 5. Layer Based Merging.

3.3. Undoing

Using *mully*, the user is allowed to create multiple views from the same graph, where views are defined by the result of the application of a set of modifications to a graph. Since the views are derived from the same graph and data, we provide the undo function in order to help the user avoid repetitive actions. Undoing helps the user to fetch the original or previous states of the *mully* graph without having to recreate it. This feature is considered the most important in this package, since it serves one of our critical aims. Undoing helps the user to document all the steps that he followed to obtain the current version of the network that he possesses. The documentation of these steps of generating views will contribute to solving the reproducibility problem, observed mostly in the research field. It will guide researchers and scientists to obtain snapshots and fragments of networks and reproduce others generated in other research.

3.4. Visualization

The *mully* package also offers a visualizer for multilayered graphs. In this visualizer, we generate layouts based on the layers, by assigning different coordinates for the nodes, where the nodes belonging to the same layer are assigned coordinated in a range of similar numbers. The user can choose between two different layouts, the random and the scaled layouts. By choosing the random layout, the nodes within a layer are displayed on random points on the display screen, while choosing a scaled layout divides the layer area display between the nodes, always making sure to avoid any overlapping vertices in both cases. We also provide the user with two visualization options: 2D and 3D visualization. The 3D visualization is generated using the rgl R package [7] which is dedicated for interactive visualization. The visualization of the same graph is shown in Figure 6 in 2D Scaled layout and in 3D.

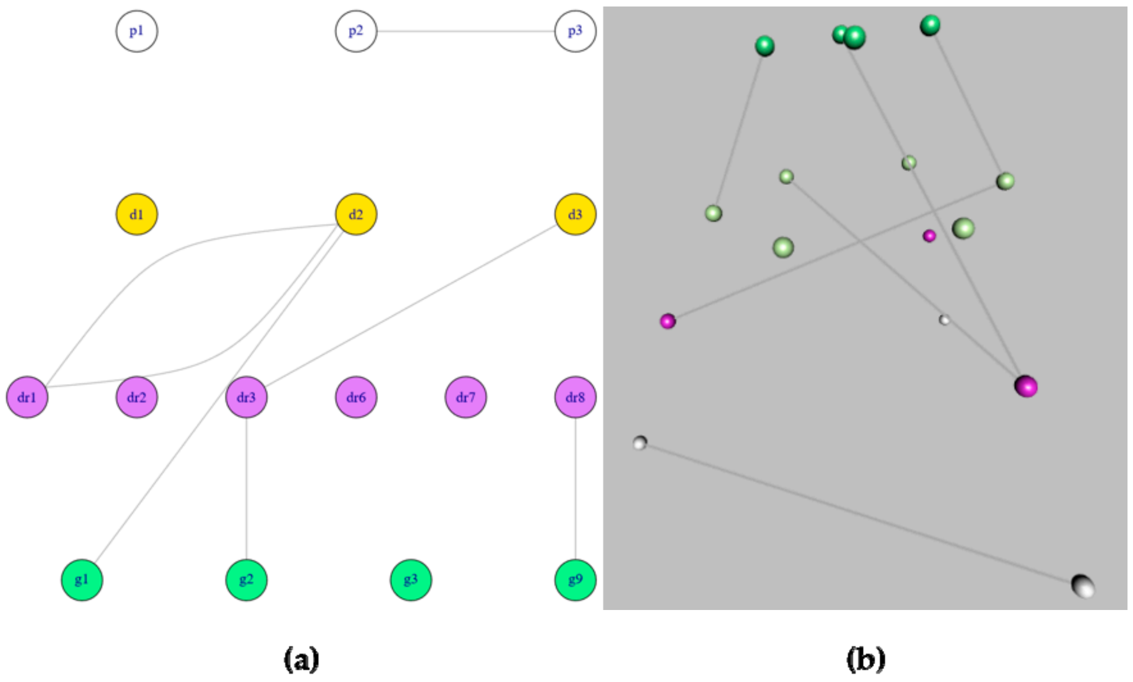


Figure 6. Visualization of the same network using the two different visualizers in *mully*. (**a**) 2D Visualization with a Scaled layout (**b**) 3D visualization.

3.5. Data Exchange

Cytoscape Cyberinfrastructure Network Interchange Format (CX) is a format for encoding network's data, developed in conjunction with the Cytoscape group [9]. It is used as a standard for network interchange by Cytoscape [10], NDEx [11], and the services in the Cytoscape Infrastructure. As this format becomes one of the standards to exchange graphs, we believe that it is essential to include it in our package. The converter provided by *mully* aims to export *mully* graphs in a CX format by using the RCX/ndexr R package [8]. By exporting the graph as a CX object, the user can then import it and use it into other tools supporting the CX format such as Cytoscape.

In this package we also provide the feature to export the graph as a CSV file. In order to export the graph, three CSV files are generated; a file containing the information about nodes, a file containing the information about edges and another for layers. This export function can also contribute in the import of a *mully* object into *mully* or other packages and tools supporting multilayered models.

4. Discussion

Multilayered graphs are currently widely used by scientists for the manipulation of big heterogeneous networks, like Social Networks, Information Networks, Technological Networks and Biological Networks [12,13]. Despite this usage, the number of tools dedicated for these graphs are still insufficient such as Arena3D [14,15], muxViz [2], etc., while we have a rich collection of tools to handle and visualize big networks on a monoplex level, of which we mention Cytoscape [10], Cell Ilustrator [16], igraph [6], Cell Designer [17], RGraphviz [18], RCytoscape [19] and many others [20].

mully is an R package that allows the user to create, modify and visualize multilayered graphs. It is implemented with special features to ease the modification and the handling of graphs by the user. It is available for free usage on Github [21].

For us, *mully* will be the stepping stone to integrate different knowledge sources and provide a reproducible knowledge network to be integrated in systems medicine approaches.

Author Contributions: Z.H. and F.K. jointly contributed to drafting, writing and editing the manuscript. Z.H. was responsible for Visualization and Software, F.K. for Funding acquisition and project administration.

Funding: This work is a part of the Multipath Project funded by the GERMAN MINISTRY OF EDUCATION AND RESEARCH (Bundesministerium für Bildung und Forschung, BMBF) grant FKZ01ZX1508.

Conflicts of Interest: The authors declare no conflict of interest.

Availability: *mully* is a free, open-source R package available for usage via Github (https://github.com/frankkramer-lab/mully).

References

1. Boccaletti, S.; Bianconi, G.; Criado, R.; Del Genio, C.I.; Gómez-Gardenes, J.; Romance, M.; Sendina-Nadal, I.; Wang, Z.; Zanin, M. The structure and dynamics of multilayer networks. *Phys. Rep.* **2014**, *544*, 1–122. [CrossRef]
2. De Domenico, M.; Porter, M.A.; Arenas, A. MuxViz: A tool for multilayer analysis and visualization of networks. *J. Complex Netw.* **2015**, *3*, 159–176. [CrossRef]
3. De Domenico, M.; Solé-Ribalta, A.; Cozzo, E.; Kivelä, M.; Moreno, Y.; Porter, M.A.; Gómez, S.; Arenas, A. Mathematical Formulation of Multi-Layer Networks. *Phys. Rev. X* **2013**, *3*, 041022.
4. Traxl, D.; Boers, N.; Kurths, J. Deep Graphs—A general framework to represent and analyze heterogeneous complex systems across scales. *Chaos* **2016**, *26*, 65303. [CrossRef] [PubMed]
5. Kivelä, M.; Arenas, A.; Barthelemy, M.; Gleeson, J.P.; Moreno, Y.; Porter, M.A. Multilayer networks. *J. Complex Netw.* **2014**, *2*, 203–271. [CrossRef]
6. Csardi, G.; Nepusz, T. The igraph software package for complex network research. *Complex Syst.* **2006**, *1695*, 1–9.
7. Murdoch, D. RGL: An R Interface to OpenGL. 2002. Available online: https://r-forge.r-project.org/projects/rgl/ (accessed on 1 February 2018).
8. Auer, F.; Hammoud, Z.; Ishkin, A.; Pratt, D.; Ideker, T.; Kramer, F. ndexr—An R package to interface with the network data exchange. *Bioinformatics* **2018**, *34*, 716–717. [CrossRef] [PubMed]
9. Available online: https://github.com/CyComponent/CyWiki/blob/master/docs/CX/CX.md (accessed on 16 January 2017).
10. Shannon, P.; Markiel, A.; Ozier, O.; Baliga, N.S.; Wang, J.T.; Ramage, D.; Amin, N.; Schwikowski, B.; Ideker, T. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Res* **2003**, *13*, 2498–2504. [CrossRef] [PubMed]
11. Pratt, D.; Chen, J.; Welker, D.; Rivas, R.; Pillich, R.; Rynkov, V.; Ono, K.; Miello, C.; Hicks, L.; Szalma, S.; et al. NDEx, the Network Data Exchange. *Cell Syst.* **2015**, *1*, 302–305. [CrossRef] [PubMed]
12. Gligorijević, V.; Pržulj, N. Methods for biological data integration: Perspectives and challenges. *J. R. Soc. Interface* **2015**, *12*, 20150571. [CrossRef] [PubMed]
13. Newman, M. The Structure and Function of Complex Networks. *SIAM Rev.* **2003**, *45*, 167–256. [CrossRef]

14. Secrier, M.; Pavlopoulos, G.A.; Aerts, J.; Schneider, R. Arena3D: Visualizing time-driven phenotypic differences in biological systems. *BMC Bioinform.* **2012**, *13*, 45–55. [CrossRef] [PubMed]
15. Pavlopoulos, G.A.; O'Donoghue, S.I.; Satagopam, V.P.; Soldatos, TG.; Pafilis, E.; Schneider, R. Arena3D: Visualization of biological networks in 3D. *BMC Syst. Biol.* **2008**, *45*, 167–256. [CrossRef] [PubMed]
16. Nagasaki, M.; Saito, A.; Jeong, E.; Li, C.; Kojima, K.; Ikeda, E.; Miyano, S. Cell Illustrator 4.0: A computational platform for systems biology. *In Silico Biol.* **2010**, *10*, 5–26. [PubMed]
17. Funahashi, A.; Matsuoka, Y.; Akiya, J.; Morohashi, M.; Kikuchi, N.; Kitano, H. Cell Designer 3.5: A versatile modeling tool for biochemical networks. *Proc. IEEE Inst. Electr. Electron. Eng.* **2008**, *96*, 1254–1265. [CrossRef]
18. Hansen, K.D.; Gentry, J.; Long, L.; Gentleman, R.; Falcon, S.; Hahne, F.; Sarkar, D. Rgraphviz: Provides Plotting Capabilities for R Graph Objects. R Package Version 2.24.0. Available online: https://www.bioconductor.org/packages/release/bioc/html/Rgraphviz.html (accessed on 17 October 2018).
19. Ono, K.; Muetze, T.; Kolishovski, G.; Shannon, P.; Demchak, B. CyREST: Turbocharging cytoscape access for external tools via a RESTful API. *F1000Research* **2015**, *4*, 478. [CrossRef] [PubMed]
20. Pavlopoulos, G.A.; Malliarakis, D.; Papanikolaou, N.; Theodosiou, T.; Enright, A.J.; Iliopoulos, I. Visualizing genome and systems biology: Technologies, tools, implementation techniques and trends, past, present and future. *Gigascience* **2015**, *4*, 38. [CrossRef] [PubMed]
21. *mully* Project on Github. Available online: https://github.com/frankkramer-lab/mully/ (accessed on 29 September 2018).

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

Vignette

mully - an R Package to create, modify and visualize multilayered graphs

true

2020-11-08

Introduction

Network theory has been used for many years in the modeling and analysis of complex systems, as epidemiology, biology and biomedicine . As the data evolves and becomes more heterogeneous and complex, monoplex networks become an oversimplification of the corresponding systems. This imposes a need to go beyond traditional networks into a richer framework capable of hosting objects and relations of different scales, called Multilayered Network **Mully**, **mul**ti**l**a**y**er networks, is an R package that provides a multilayer network framework. Using this package, the user can create, modify and visualize graphs with multiple layers. This package is an extension to the igraph package that provides a monolayer graph framework. The package is implemented as a part of the Multipath Project directed by Dr. Frank Kramer . ## Publication More information and references can be found in the mully paper:

https://www.mdpi.com/2073-4425/9/11/519

Installation

Installation via Github

```
require(devtools)
install_github("frankkramer-lab/mully")
```

Load the package

```
library("mully")
#> Loading required package: igraph
#>
#> Attaching package: 'igraph'
#> The following objects are masked from 'package:stats':
#>
#>     decompose, spectrum
#> The following object is masked from 'package:base':
#>
#>     union
#> Loading required package: rgl
#> Loading required package: randomcoloR
#> Loading required package: shape
```

```
#>
#> Attaching package: 'mully'
#> The following object is masked from 'package:rgl':
#>
#>     plot3d
#> The following object is masked from 'package:utils':
#>
#>     demo
#> The following object is masked from 'package:base':
#>
#>     merge
```

Test the package

In this section, we provide a demo to test the package by calling some of the function. After running this script, you will have a graph g with 3 layers and 8 nodes. the graph can also be modified by calling other functions. Please refer to help to see the available functions. ### Create new mully graph

```
g=mully("MyFirstMully",direct = F)
```

Add Layers

```
g=addLayer(g, c("Gene", "Drug", "Disease"))
```

Add/print Nodes

```
g=addNode(g,"d1","disease",attributes=list(type="t1"))

g=addNode(g,"d2","disease",attributes=list(type="t1"))

g=addNode(g,"d3","disease",attributes=list(type="t1"))

g=addNode(g,"dr1","drug",attributes=list(effect="strong"))

g=addNode(g,"dr2","drug",attributes=list(effect="strong"))

g=addNode(g,"dr3","drug",attributes=list(effect="moderate"))

g=addNode(g,"g1","gene",attributes=list(desc="AF"))

g=addNode(g,"g2","gene",attributes=list(desc="BE"))

#See vertices attributes
print(getNodeAttributes(g))
```

Add/print/remove Edges

```
g=addEdge(g,"dr1","d2",list(name="treats"))
g=addEdge(g,"dr1","d2",list(name="extraEdge"))
g=addEdge(g,"d2","g1",list(name="targets"))
g=addEdge(g,"g2","dr3",list(name="mutates and causes"))
g=addEdge(g,"dr3","d3",list(name="treats"))

print(getEdgeAttributes(g))

removeEdge(g,"d2","dr1",multi=T)
```

Merge two graphs

```
#Create a Second graph
g1=mully()

g1=addLayer(g1,c("protein","drug","gene"))

g1=addNode(g1,"dr4","drug",attributes=list(effect="strong"))
g1=addNode(g1,"dr5","drug",attributes=list(effect="strong"))
g1=addNode(g1,"dr6","drug",attributes=list(effect="moderate"))

g1=addNode(g1,"p1","protein")
g1=addNode(g1,"p2","protein")
g1=addNode(g1,"p3","protein")

g1=addNode(g1,"g3","gene")
g1=addNode(g1,"g4","gene")

g1=addEdge(g1,nodeStart = "p2",nodeDest = "p3",attributes = list(name="interacts"))
g1=addEdge(g1,nodeStart = "dr6",nodeDest = "g4",attributes = list(name="targets"))

#Merge both graphs
g12=merge(g,g1)

#Print the graph
print(g12)
```

Visualization

```
plot(g12,layout = "scaled")
```

```
plot3d(g12)
```

Available Functions

mully functions are divided into different files depending on their functionnality range: Constructor , Layers Functions , Node Functions , Edge Functions , Merge Function , Visualization Functions , Import Functions

, Export Functions , Demo.

Function	Description
`mully(name,direct)`	Constructor Function, Create an empty multilayered graph
`print(g)`	Print function
`addLayer(g, nameLayer)`	Add a layer or a set of layers to a graph
`removeLayer(g, name,trans)`	Delete a layer or a set of layers from a graph
`isLayer(g, name)`	Verify if the layer exists in a graph
`getLayersCount(g)`	Get the number of layers in a graph
`getLayer(g, nameLayer)`	Get the nodes on a layer in a graph
`getNode(g,nameNode)`	Get a node from a graph
`getIDNode(g,nameNode)`	Get the id of a node
`addNode(g, nodeName, layerName, attributes)`	Add a node with assigned layer and attributes to a graph
`removeNode(g, name,trans)`	Delete a node or a set of nodes from a graph
`getNodeAttributes(g,nameNode)`	Get the attributes of one or all nodes
`addEdge(g, nodeStart, nodeDest, attributes)`	Add an edge
`removeEdge(g, nodeStart, nodeDest,attributes, multi)`	Delete an edge
`getEdgeAttributes(g,nodeStart,nodeDest)`	Get the attributes of the edges connecting two nodes or all the edges in the graph
`getIDEdge(g,nodeStart,nodeDest)`	Get the ids of the edges connecting two nodes
`merge(g1,g2)`	Merge or unite two graphs
`plot(g,layout)`	Plot the graph in 2D
`plot3d(g)`	Plot the graph in 3D using rgl
`importGraphCSV(name,direct,layers,nodes,edges)`	Import a mully graph from csv files
`importLayersCSV(g,file)`	Import layers to a mully graph from a CSV file
`importNodesCSV(g,file)`	Import nodes to a mully graph from a CSV file
`importEdgesCSV(g,file)`	Import edges to a mully graph from a CSV file

addEdge

Add an edge

The function is used to add an edge to a given mully graph. mully supports multi-edges, i.e. two nodes can be connected with multiple edges. The uniqueness of the edges is based on the connection with the attributes. When adding a new connection between two nodes that are already cnnected, the new edge should have different attributes. The function needs the following arguments:

- **g** - The input graph
- **nodeStart** - The first endpoint of the edge
- **nodeDest** - The second endpoint of the edge
- **attributes** - A named list, the attributes to assign to the edge

The function returns the graph, with the added edge.

Example

```
g=demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
addEdge(g,"dr3","g2",attributes=list(name="newEdge"))
#> mully --  MyFirstMully
#>  3 Layers:
#>   ID    Name NameLower
#>  1  1    Gene      gene
#>  2  2    Drug      drug
#>  3  3 Disease   disease
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
#>
#>  5 Edges:
#>    V1  V2               name
#>  1  d2 dr1             treats
#>  2  d2 dr1          extraEdge
#>  3  d2  g1            targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3             treats
```

addLayer

Add a layer or a set of layers to a graph

The function is used to add a new layer to a given mully graph. The layer name should be unique and should not already exist in the graph. The names are not case-sensitive. the function has an internal count, and assigns new IDs to new layers. The internal counr number only increases, i.e. it does not change after deleting a lyer. Therefore the ID of the last added layer is not necessarily the number of layers in the mully graph. The function needs the following arguments:

- **g -** The input graph
- **nameLayer -** The name or the list of the names of the layers to be added. The layer names must be unique

The function returns the graph, with the layers added.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
addLayer(g,"Complex")
#> mully --  MyFirstMully
#>  4 Layers:
#>   ID    Name NameLower
#>  1  1    Gene      gene
#>  2  2    Drug      drug
#>  3  3 Disease   disease
#>  4  4 Complex   complex
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
#>
#>  5 Edges:
#>    V1  V2               name
#>  1  d2 dr1             treats
#>  2  d2 dr1          extraEdge
#>  3  d2  g1            targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3             treats
```

addNode

Add a node with assigned layer and attributes to a graph

The function is used to add a node to a given mully graph. The layer to which this node should be added is required. Nodes on single layers should be unique, but can not be on different layers. Layer Names are not case-sensitive. The function needs the following arguments:

- **g** - The input graph
- **nodeName** - The name of the node to add
- **layerName** - The name of the layer to be assigned to the node
- **attributes** - The attributes of the node to add. This argument must be a named list

The function returns the graph, with the new node.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
attributes=list("specie"="Homo Sapiens")
addNode(g = g,nodeName = "g3",layerName = "Gene",attributes = attributes)
#> mully --  MyFirstMully
#>  3 Layers:
#>   ID    Name NameLower
#>  1  1    Gene      gene
#>  2  2    Drug      drug
#>  3  3 Disease   disease
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
#>
```

```
#>  5 Edges:
#>    V1  V2               name
#>  1  d2 dr1            treats
#>  2  d2 dr1         extraEdge
#>  3  d2  g1           targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3            treats
```

exportCSV

Export mully into CSV files

The function is used to export a given mully graph in the CSV Format. Three files will be generated, respectively containing the layers', nodes' and edges' information. The function needs the following arguments:

- **g** - The input graph
- **target** - The target file in which the files will be generated. By default the WD.

Example

```
g=mully::demo()
exportCSV(g)
```

getEdgeAttributes

Get the attributes of the edges connecting two nodes

The function is used to obtain information on the edges in a given mully graph. The information can be on a single edge or all the edges in the graph. The function needs the following arguments:

- **g** - The input graph
- **nodeStart** - The first endpoint of the edge
- **nodeDest** - The second endpoint of the edge

The function returns a dataframe containing the edges with their attributes. If both nodes' arguments are missing, it returns the complete information on edges and their attributes.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2              name
#> 1  d2 dr1            treats
#> 2  d2 dr1           extraEdge
#> 3  d2  g1             targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3              treats
##Print all Edges
getEdgeAttributes(g)
#>    V1  V2                name
#> 1  d2 dr1              treats
#> 2  d2 dr1           extraEdge
#> 3  d2  g1             targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3              treats
##Get a Single Edge
getEdgeAttributes(g,"d2","g1")
#>   V1 V2    name
#> 3 d2 g1 targets
```

getIDEdge

Get the ids of the edges connecting two nodes

The function is used to get the internal ID of an edge in a given mully graph. The function needs the following arguments:

- **g** - The input graph
- **nodeStart** - The first endpoint of the edge
- **nodeDest** - The second endpoint of the edge

The nodes passed to this function as arguments should be the unique names assigned to the nodes upon addition. mully supports multi-edges, therefore this function may return a list of edges connecting two nodes. The function returns a list containing the ids of the edges connecting the nodes.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2                name
#> 1  d2 dr1              treats
#> 2  d2 dr1           extraEdge
#> 3  d2  g1             targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3              treats
getIDEdge(g,"d2","dr1")
#> [1] 1 2
```

getIDNode

Get the id of a node

The function is used to get the internal ID of a given node in a given mully graph. The function needs the following arguments:

- **g -** The input graph
- **nameNode -** The name of the node

The function returns the id of the specified node.

getLayer

Get the nodes on a layer in a graph

The function is used to get the nodes on a given layer in a given mully graph. The layer name is not case-sensitive. The function needs the following arguments:

- **g -** The input graph
- **nameLayer -** The name of the layer

The function returns a List of the nodes on the given layer.

Example

```r
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
getLayer(g,"gene")
#> + 2/8 vertices, named, from aa24447:
#> [1] g1 g2
```

getLayersCount

Get the number of layers in a graph

The function is used to get the number of layers in a given mully graph. The function needs the following arguments:

- **g** - The input graph

The function returns the count of the layers.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2                name
#> 1  d2 dr1              treats
#> 2  d2 dr1           extraEdge
#> 3  d2  g1             targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3              treats
getLayersCount(g)
#> [1] 3
```

getNode

Get a node from a graph

The function is used to get a node from a given mully graph as an igraph.vs object. The function needs the following arguments:

- **g** - The input graph
- **name** - The name of the node

The function returns the node as igraph.vs.

getNodeAttributes

Get the attributes of a node

The function is used to get a node or a list of nodes with corresponding attributes from a given mully graph. If the node name is missing, the complete node information is extracted. The function needs the following arguments:

- **g** - The input graph
- **nameNode** - The name of the node
- **layerByName** - A boolean to specify whether to export the layers by name or by ID

The function returns a dataframe containing the attributes of the specified node.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
getNodeAttributes(g,layerByName = TRUE)
#>   name       n type   effect desc
#> 1   d1 Disease   t1     <NA> <NA>
#> 2   d2 Disease   t1     <NA> <NA>
#> 3   d3 Disease   t1     <NA> <NA>
#> 4  dr1    Drug <NA>   strong <NA>
#> 5  dr2    Drug <NA>   strong <NA>
#> 6  dr3    Drug <NA> moderate <NA>
#> 7   g1    Gene <NA>     <NA>   AF
#> 8   g2    Gene <NA>     <NA>   BE
```

importEdgesCSV

Import Edges to a mully graph from a CSV file

The function is used to import edges to a mully graph. The function reads a CSV file and adds edges between existing nodes to the graph. The graph should already contain layers and nodes. The function needs the following arguments:

- **g** - The mully graph to which the nodes will be added. The graph should already have the layers and the nodes.
- **file** - The path to the CSV file containing the edges' information

The function returns the mully graph with the added edges.

importGraphCSV

Import a mully graph from CSV files

The function is used to create a graph from CSV files. To create the graph, three files are needed: the layers, nodes and edges. mully also offers functions to import individual files containing nodes', edges' and layers information. See Functions `importLayersCSV()`, `importNodesCSV()` and `importEdgesCSV()`. The function needs the following arguments:

- **name** - The name of the graph

- **direct** - A boolean to indicate if the graph is directed or not
- **layers** - The path to the CSV file containing the layers' information
- **nodes** - The path to the CSV file containing the nodes' information
- **edges** - The path to the CSV file containing the edges' information

The function returns a new mully graph.

importLayersCSV

Import Layers to a mully graph from a CSV file

The function is used to add layers to a mully graph. The function reads a CSV file and adds the layers to the graph. The file should contain the layers' names. Layer IDs are assigned automatically. The function needs the following arguments:

- **g** - The mully graph to which the layers will be added. If missing, a new mully graph is created
- **file** - The path to the CSV file containing the layers' information

The function returns the mully graph with the added layers.

importNodesCSV

Import Nodes to a mully graph from a CSV file

The function is used to import nodes into an existing mully graph. The function reads a CSV file and adds the nodes with the attributes to the graph. The graph should already contain layers, and the nodes in the file should have an attribute n referring to the layer assignment. The function needs the following arguments:

- **g** - The mully graph to which the nodes will be added. The graph should already have the layers
- **file** - The path to the CSV file containing the nodes' information
- **name** - The name of the column containing the names of the nodes

The function returns the mully graph with the added nodes.

is.mully

Is this a mully graph?

The function check if a given graph is a mully graph. The function needs the following arguments:

- **g** - The input graph

The function returns a boolean indicating whether the graph is or not a mully object. ### isLayer **Verify if the layer exists in a graph**

This function is used to check if a given mully graph contains a given layer. The layer name is not case-sensitive. The function needs the following arguments:

- **g** - The input graph
- **name** - The name of the layer

The function returns a boolean value.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
isLayer(g,"Gene")
#> [1] TRUE
isLayer(g,"Complex")
#> [1] FALSE
```

merge

Merge or unite two graphs

The function is used to merge two mully graphs. The merge is layer based, i.e. nodes belonging to similar layers in both graphs will be combined in the returned graph. The node names are unique on single layers but can be redundant over different layers. The merge is based on the first arguments, therefore all elements of the second argument will be added to the first. The function needs the following arguments:

- **g1** - The first graph to merge. This is the base of the merge.
- **g2** - The second graph to merge. All of its elements are added to the first graph.

The function returns the merge of the two graphs. The merge is based on the first given graph.

Example

```
#Create First Graph
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats

#Create Second Graph
g1 <- mully("MySecondMully",direct = F)

g1 <- addLayer(g1, c("gene", "Protein", "Drug"))

g1=addNode(g1,"p1","Protein",attributes=list(type="t1"))

g1=addNode(g1,"p2","Protein",attributes=list(type="t1"))

g1=addNode(g1,"p3","Protein",attributes=list(type="t1"))

g1=addNode(g1,"dr6","drug",attributes=list(effect="strong"))

g1=addNode(g1,"dr7","drug",attributes=list(effect="strong"))

g1=addNode(g1,"dr8","drug",attributes=list(effect="moderate"))

g1=addNode(g1,"g3","gene",attributes=list(desc="AF"))

g1=addNode(g1,"g9","gene",attributes=list(desc="BE"))

g1=addEdge(g1,"dr8","g9",list(name="targets"))

g1=addEdge(g1,"p3","p2",list(name="interactWith"))

#Merge Graphs
merge(g,g1)
#> Warning in addLayer(g1, g2$layers$Name): Layer gene Already Exists and will be
#> skipped
#> Warning in addLayer(g1, g2$layers$Name): Layer Drug Already Exists and will be
#> skipped
#> mully --  MyFirstMully
#>  4 Layers:
#>   ID    Name NameLower
#>  1  1    Gene      gene
#>  2  2    Drug      drug
#>  3  3 Disease   disease
#>  4  4 Protein   protein
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
```

```
#>
#>  5 Edges:
#>    V1  V2               name
#>  1  d2 dr1             treats
#>  2  d2 dr1          extraEdge
#>  3  d2  g1            targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3             treats
```

mully

Create an empty multilayered graph

The function create an empty mully graph. The function needs the following arguments:

- **name** - The name to be assigned to the graph.
- **direct** - A boolean value, if the graph is directed or not. By default TRUE.

The function returns the created multilayered graph. ### plot,mully **Plot the graph in 2D**

The function is used to plot the mully graph. It uses the plot function from igraph. We do not recommend using this function to plot big graphs, and use plot3d instead. The function needs the following arguments:

- **g** The input graph
- **layout** The layout. Can either be random or scaled

plot3d

Plot the graph in 3D using rgl

The funstion is used to generate 3D interactive plots of a mully graph. The 3D plot is generated using the R Package rgl, and uses some of the arguments passed to the igraph function rgl.plot. Most of the arguments are set to default values, but can be changed. The layout is calculated internally based on the layers. The function needs the following arguments:

- **g** The input graph
- **layers** A boolean whether to add the layers or not
- **vertex.label** The vertices' labels
- **vertex.label.color** The vertices' colors. If not specified, the colors will be chosen randomly
- **vertex.plac** The placement form of the vertices on the layer. Can either be "circle" which will place them on a circle, or "disc" which will place them randomly on a disc. The default is "circle"
- **edge.color** The edges' colors.If not specified, inter-edges are black, and intra-edges have the same color as the nodes on the layer
- **edge.width** The edge width. Default set to 5.
- **edge.arrow.size** The edges' arrow size. Default set to 10
- **edge.arrow.width** The edges' arrow width. Default set to 1

The function can take the following arguments supported and not ignored by rglplot}: vertex.label, vertex.label.color, edge.color, edge.width, edge.arrow.size,edge.arrow.width.

print,mully

Print a mully graph

The function prints a mully graph. The function needs the following arguments:

- **g** - The input graph

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
print(g)
#> mully --  MyFirstMully
#>  3 Layers:
#>   ID    Name NameLower
#>  1  1    Gene      gene
#>  2  2    Drug      drug
#>  3  3 Disease   disease
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
#>
#>  5 Edges:
#>    V1  V2               name
#>  1  d2 dr1             treats
#>  2  d2 dr1          extraEdge
#>  3  d2  g1            targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3             treats
```

removeEdge

Delete an edge

The function is used to remove an edge from a given mully graph. Since mully supports multi-edges, a boolean *multi* is needed to specify whether to delete multiple edges or a single edge. In case one edge should be deleted, the named list of attributes of the edge is required. The function needs the following arguments:

- **g** - The input graph
- **nodeStart** - The first endpoint of the edge
- **nodeDest** - The second endpoint of the edge
- **attributes** - The attributes of the edge to delete. Required if the nodes are multi-connected
- **multi** - A boolean. Specifies whether to delete multiple edges or not, in case they exist.

The function returns the graph with the deleted edges.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
removeEdge(g,"dr1","d2",multi=TRUE)
#> mully --  MyFirstMully
#>  3 Layers:
#>   ID    Name NameLower
#>  1  1    Gene      gene
#>  2  2    Drug      drug
#>  3  3 Disease   disease
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
#>
```

```
#>  5 Edges:
#>    V1  V2               name
#>  1  d2 dr1             treats
#>  2  d2 dr1          extraEdge
#>  3  d2  g1            targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3             treats
```

removeLayer

Delete a layer or a set of layers from a graph

The function is used to remove a layer from a given mully graph. Removing a layer results deleting all nodes assigned to the layer, and the edges connecting these nodes to other nodes in the graph. Transitive edges between nodes can be added by setting trans to TRUE. The function needs the following arguments:

- **g** - The input graph
- **name** - The name or the list of the names of the layers to be deleted
- **trans** - A boolean whether to insert transitive edges or not

The function returns the graph, with the given layer and its corresponding nodes and edges removed.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2               name
#> 1  d2 dr1             treats
#> 2  d2 dr1          extraEdge
#> 3  d2  g1            targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3             treats
removeLayer(g,"gene",trans=TRUE)
#> mully --  MyFirstMully
#>  3 Layers:
#> [1] ID         Name      NameLower
#>  <0 rows> (or 0-length row.names)
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
#>
#>  5 Edges:
#>    V1  V2                name
#>  1  d2 dr1              treats
#>  2  d2 dr1           extraEdge
#>  3  d2  g1             targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3              treats
```

removeNode

Delete a node or a set of nodes from a graph

The function is used to remove a node from a given mully graph. Removing a node results removing the edges connecting this node to other nodes in the graph. Transitive edges can also be added after deletion by setting trans to TRUE (a->b, b->c, removing b with trans=TRUE will add a->c). The function needs the following arguments:

- **g** - The input graph
- **name** - The name or the list of names of the nodes to be deleted
- **trans** - A boolean whether to insert transitive edges or not

The function returns the graph, with the nodes deleted.

Example

```
g=mully::demo()
#> Warning in addLayer(g, c("Gene", "Drug", "Drug", "Disease")): Layer Drug Already
#> Exists and will be skipped
#> [1] "Node d1 added as disease"
#> [1] "Node d2 added as disease"
#> [1] "Node d3 added as disease"
#> [1] "Node dr1 added as drug"
#> [1] "Node dr2 added as drug"
#> [1] "Node dr3 added as drug"
#> [1] "Node g1 added as gene"
#> [1] "Node g2 added as gene"
#>    V1  V2                name
#> 1  d2 dr1              treats
#> 2  d2 dr1           extraEdge
#> 3  d2  g1             targets
#> 4 dr3  g2 mutates and causes
#> 5  d3 dr3              treats
removeNode(g,"dr1",trans=TRUE)
#> mully --  MyFirstMully
#>  3 Layers:
#>   ID    Name NameLower
#>  1  1    Gene      gene
#>  2  2    Drug      drug
#>  3  3 Disease   disease
#>
#>  8 Nodes:
#>   name n type   effect desc
#>  1   d1 3   t1     <NA> <NA>
#>  2   d2 3   t1     <NA> <NA>
#>  3   d3 3   t1     <NA> <NA>
#>  4  dr1 2 <NA>   strong <NA>
#>  5  dr2 2 <NA>   strong <NA>
#>  6  dr3 2 <NA> moderate <NA>
#>  7   g1 1 <NA>     <NA>   AF
#>  8   g2 1 <NA>     <NA>   BE
#>
#>  5 Edges:
#>    V1  V2               name
#>  1  d2 dr1              treats
#>  2  d2 dr1           extraEdge
#>  3  d2  g1             targets
#>  4 dr3  g2 mutates and causes
#>  5  d3 dr3              treats
```

Appendix C

Publication “Multipath: an R Package to generate integrated reproducible pathway models”

biology

Technical Note

Multipath: An R Package to Generate Integrated Reproducible Pathway Models

Zaynab Hammoud * and Frank Kramer

IT-Infrastructure for Translational Medical Research, University of Augsburg, 86159 Augsburg, Germany; frank.kramer@informatik.uni-augsburg.de

* Correspondence: zaynab.hammoud@informatik.uni-augsburg.de or zaynabhassanhammoud@gmail.com

Received: 24 October 2020; Accepted: 19 December 2020; Published: 21 December 2020

Simple Summary: In biological terms, the term "pathway" is used to describe a collection of processes within a cell that lead to one or more actions. The graphical representation of these processes enables the reader to understand complex relationships and interactions much more easily compared to free-text descriptions. While there is usually agreement on the existence and function of these high-level processes, the specific molecules and their interactions are often disputed and a matter of current research. A standardized computational representation of biological networks has become desirable, especially with the recent surge in new knowledge generation in biology and medicine. Our work is influenced by challenges emerging from previous work on biological pathways, knowledge encoding, and visualization as well as pathway databases. Our main motivation is the difficulty of reproducing pathway knowledge used within publications, even in top-tier journals. We propose a new way of integrating and modeling pathways and other influencing knowledge, such as drugs, and documenting their modifications using multilayered networks. We provide a tool that transforms encoded pathway data to multilayered graphs, with the possibility to modify them, and integrate other knowledge from external databases.

Abstract: Biological pathway data integration has become a topic of interest in the past years. This interest originates essentially from the continuously increasing size of existing prior knowledge as well as from the many challenges scientists face when studying biological pathways. Multipath is a framework that aims at helping re-trace the use of specific pathway knowledge in specific publications, and easing the data integration of multiple pathway types and further influencing knowledge sources. Multipath thus helps scientists to increase the reproducibility of their code and analysis by allowing the integration of numerous data sources and documentation of their integration steps while doing so. In this paper, we present the package Multipath, and we describe how it can be used for data integration and tracking pathway modifications. We present a multilayer model built from the Wnt Pathway as a demonstration.

Keywords: multilayer networks; data integration; biological pathways; reproducibility; visualization

1. Introduction

Deciphering the inner workings of cells fascinates researchers all over the world. However, these processes are highly complex and heterogeneous, with countless players constantly interacting via biochemical reactions, signaling cascades and feedback loops. Knowledge about these processes can be organized into so-called pathways by grouping sets of interactions, which share a common goal or function [1]. Over the course of the last decades, an enormous amount of knowledge on molecular interactions within cells has been accumulated, with a plethora of methods and algorithms using this knowledge. Consequently, two problems are currently faced: the integration of the different pathways' types and the irreproducibility of pathway illustrations used within journals.

Our aim is to provide a generic modelling framework to integrate multiple pathway types and further knowledge sources influencing these pathways. This framework is defined by a multi-layered model allowing automatic pathway transformations and documentation. By providing a tool that generates this model, we aim to facilitate the data integration, boost the reproducibility and increase the interoperability between different sources and databases in the field of pathways. The transformation of the pathway models helps structuring and condensing knowledge in specific areas of interests, e.g., removing layers with irrelevant information, coupling expression and pathway knowledge, and connecting drug-target and mutation layer connections, while documenting every step of this process.

In this paper, we present the R package Multipath that creates these multilayer extendable models from BioPAX-encoded pathway files [2], and extract influencing knowledge from external databases. We show the multilayer model and the views that we generated from the Signaling by Wnt Signaling pathway.

This package uses freely available data sources like Reactome or other ontology-encoded pathway databases, using our R Packages rBiopaxParser [3] and mully [4]. Furthermore, we integrate open source tools like dbparser [5] and UniProt.ws [6] to retrieve information from the databases DrugBank [7] and UniProt [8].

2. Materials and Methods

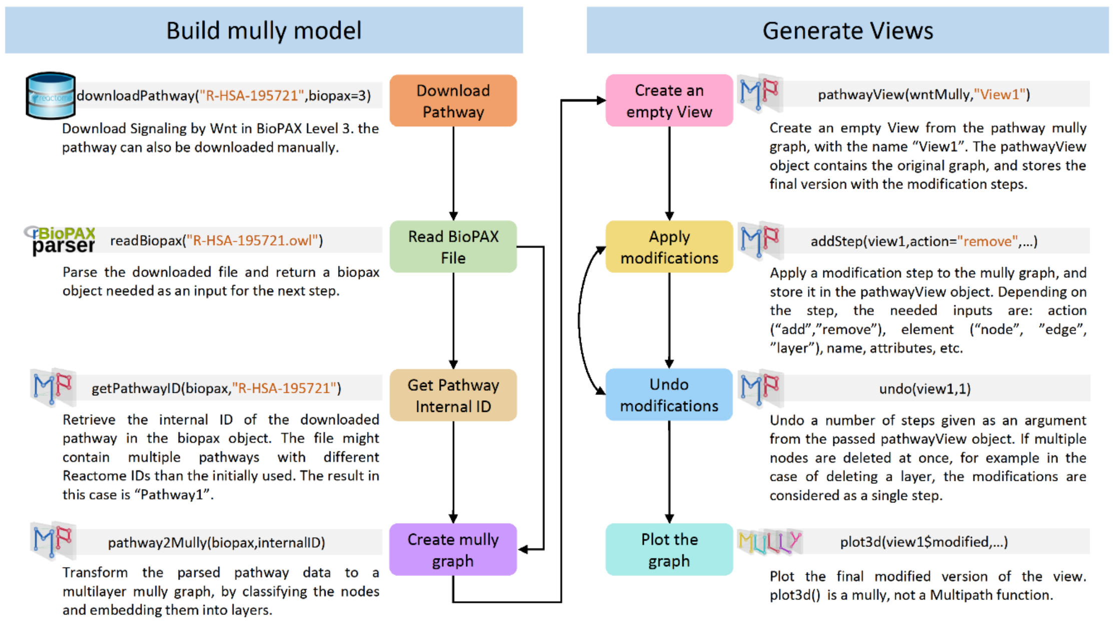
Multipath is an R package that aims at creating reproducible pathway models. It allows the user to transform BioPAX encoded pathways into multilayered graphs, using the R packages mully [4] and rBiopaxparser [3]. Mully is an R package that allows the creation, modification, and visualization of multilayered graphs [4], while rBiopaxparser is an R package to parse, modify, and visualize pathway data encoded in the BioPAX standard [3]. It allows the user to parse the data and create a monolayered graph from it. The package mully is then used to transform this monolayered graph into a multilayered mully model. The elements of the pathways—which represent the nodes—are divided into groups based on their class, for example complexes, proteins, DNAs, RNAs, small molecules, etc. Each group of nodes is embedded into one layer of the resulting graph. The edges connecting the nodes are the interactions extracted from the BioPAX file.

To build the BioPAX mully model, the user has to follow the following steps (Figure 1):

1. Read the BioPAX file using the rBiopaxparser function readBiopax() and provide the path to the BioPAX file containing the pathway information as an argument. The file can be downloaded manually or using the function downloadPathway().
2. Fetch the pathway's internal ID using the function getPathwayID() from the BioPAX object returned in step 1 which might contain multiple pathways.
3. Provide the pathway's ID alongside the BioPAX object as arguments to the Multipath function pathway2Mully() which returns the mully graph created from the parsed pathway information in the BioPAX object.

The processing time depends on the size of the BioPAX file being imported. A larger pathway may require a larger memory space and a longer time to be parsed, and transformed into a mully object.

We currently offer the function downloadPathway(), to download a pathway from the database Reactome [9] in BioPAX level 2 and 3. However, the user can download the pathways manually. A list of available repositories for pathways encoded in the BioPAX standard can be found in the pathway resource list Pathguide [10].



```
downloadPathway("R-HSA-195721",biopax=3)
```

Download Signaling by Wnt in BioPAX Level 3. the pathway can also be downloaded manually.

```
readBiopax("R-HSA-195721.owl")
```

Parse the downloaded file and return a biopax object needed as an input for the next step.

```
getPathwayID(biopax,"R-HSA-195721")
```

Retrieve the internal ID of the downloaded pathway in the biopax object. The file might contain multiple pathways with different Reactome IDs than the initially used. The result in this case is "Pathway1".

```
pathway2Mully(biopax,internalID)
```

Transform the parsed pathway data to a multilayer mully graph, by classifying the nodes and embedding them into layers.

```
pathwayView(wntMully,"View1")
```

Create an empty View from the pathway mully graph, with the name "View1". The pathwayView object contains the original graph, and stores the final version with the modification steps.

```
addStep(view1,action="remove",...)
```

Apply a modification step to the mully graph, and store it in the pathwayView object. Depending on the step, the needed inputs are: action ("add","remove"), element ("node", "edge", "layer"), name, attributes, etc.

```
undo(view1,1)
```

Undo a number of steps given as an argument from the passed pathwayView object. If multiple nodes are deleted at once, for example in the case of deleting a layer, the modifications are considered as a single step.

```
plot3d(view1$modified,...)
```

Plot the final modified version of the view. plot3d() is a mully, not a Multipath function.

Figure 1. Reproducibility Workflow Diagram. The diagram shows the steps that have to be followed to generate reproducible multilayer pathway models. The Signaling by Wnt pathway was used as an example.

The generated mully graph is modifiable and retraceable. All modifications applied to the model can be stored in a view object and tracked using the feature track and undo. The view object contains different information concerning the modifications, including the timestamp of creation and last modification, the original graph and its final modified version, as well as the list of applied steps.

To track the modification, the user has to call the function addStep(), to which a list of arguments should be provided: the action (add or remove), the element (node, edge or layer), the name of the element, and more arguments depending on the type of the element. It returns the view with the added steps. Removing a layer results removing a list of nodes and edges, which are all considered a single step.

The function undo(), which reverses the latest changes applied to the graph, requires the view and the number of steps to undo. The steps are reversed based on their id in the view. For instance, undoing a deletion of a node, will re-add the node and all its connections, and of a layer will re-add the layer, the nodes of this layer, and all their connections.

Multipath also offers data integration functions to extract any additional information needed from DrugBank [7] and UniProt [6]. These functions, that need the list of proteins' and drugs' IDs respectively as input, return integrated information existing in both databases.

Functions querying other databases such as ChEBI [11] and PubChem [12] using the R Package webchem [13], KEGG [14], and ENSEMBL [15] are also being implemented and will be available soon.

3. Results

3.1. Pathway Data Integration

We present functions in Multipath that fetch information from external databases. Essential information on each node are taken from the original database of the given IDs; for example, information on proteins with UniProtKB/Swiss-Prot IDs are extracted from UniProt. The interactions are proved from different databases, and the source of the interactions added to the graph is assigned as an attribute to the edges.

3.1.1. DrugBank

DrugBank is an online freely accessible database for drug data and drug products [7]. The complete data can be downloaded from the official DrugBank website [16] in the XML format. To parse this data, the function loadDBXML() uses the R package dbparser [5]. This function should be called before using any other function related to DrugBank, since it returns the object containing the parsed information, and needed as an argument in all the related functions (Figure 2). The parsed data contain information on the drugs, as well as interactions between the drugs and between drugs and proteins. The latter is divided into four types: transporters, carriers, enzymes, and targets. All four types are combined and arranged into a protein layer, which will be connected to the drug layer. The type is added as an attribute to the nodes.

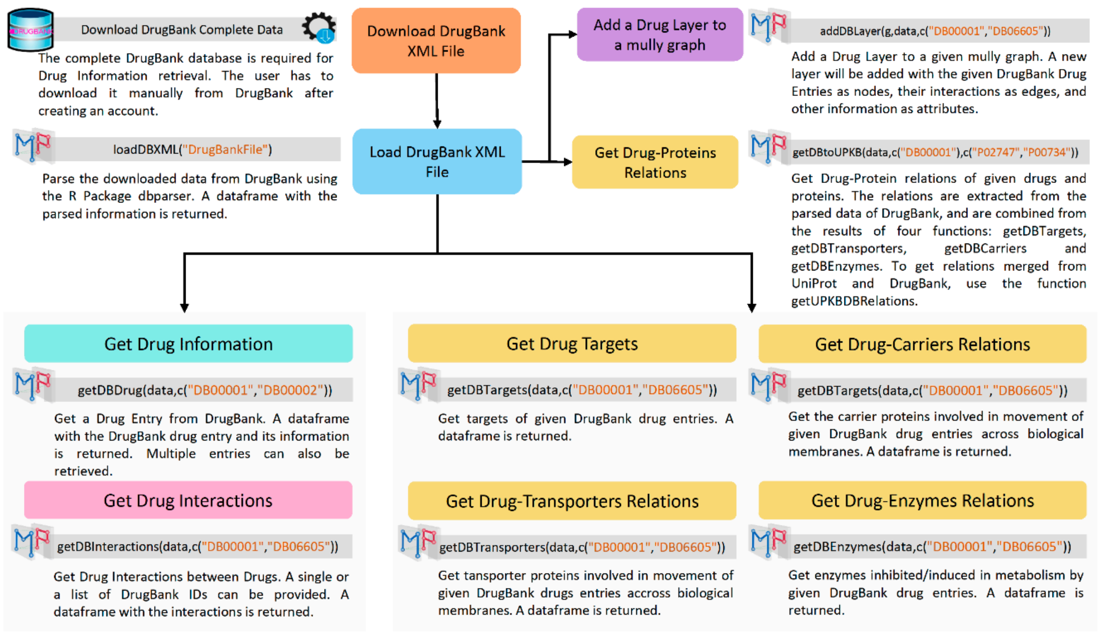

Download DrugBank Complete Data
The complete DrugBank database is required for Drug Information retrieval. The user has to download it manually from DrugBank after creating an account.
loadDBXML("DrugBankFile")
Parse the downloaded data from DrugBank using the R Package dbparser. A dataframe with the parsed information is returned.
Download DrugBank XML File
Load DrugBank XML File
Add a Drug Layer to a mully graph
addDBLayer(g,data,c("DB00001","DB06605"))
Add a Drug Layer to a given mully graph. A new layer will be added with the given DrugBank Drug Entries as nodes, their interactions as edges, and other information as attributes.
Get Drug-Proteins Relations
getDBtoUPKB(data,c("DB00001"),c("P02747","P00734"))
Get Drug-Protein relations of given drugs and proteins. The relations are extracted from the parsed data of DrugBank, and are combined from the results of four functions: getDBTargets, getDBTransporters, getDBCarriers and getDBEnzymes. To get relations merged from UniProt and DrugBank, use the function getUPKBDBRelations.
Get Drug Information
getDBDrug(data,c("DB00001","DB00002"))
Get a Drug Entry from DrugBank. A dataframe with the DrugBank drug entry and its information is returned. Multiple entries can also be retrieved.
Get Drug Interactions
getDBInteractions(data,c("DB00001","DB06605"))
Get Drug Interactions between Drugs. A single or a list of DrugBank IDs can be provided. A dataframe with the interactions is returned.
Get Drug Targets
getDBTargets(data,c("DB00001","DB06605"))
Get targets of given DrugBank drug entries. A dataframe is returned.
Get Drug-Carriers Relations
getDBTargets(data,c("DB00001","DB06605"))
Get the carrier proteins involved in movement of given DrugBank drug entries across biological membranes. A dataframe is returned.
Get Drug-Transporters Relations
getDBTransporters(data,c("DB00001","DB06605"))
Get tansporter proteins involved in movement of given DrugBank drugs entries accross biological membranes. A dataframe is returned.
Get Drug-Enzymes Relations
getDBEnzymes(data,c("DB00001","DB06605"))
Get enzymes inhibited/induced in metabolism by given DrugBank drug entries. A dataframe is returned.

Figure 2. DrugBank Workflow Diagram. The Diagram shows the steps to get Drug Information from DrugBank using Multipath and the R Package dbparser.

After calling loadDBXML(), the functions getDBDrug() and getDBInteractions() can be called to fetch respectively the drug entries' information and the interactions between them from DrugBank, using the parsed DrugBank object, and the list of DrugBank IDs. The addition of a Drug Layer to an existing mully graph can be achieved using the function addDBLayer(), with the mully graph, the parsed DrugBank object, and the list of DrugBank IDs as arguments.

The connections to the protein nodes on the protein Layer are returned by the function getDBtoUPKB(), which combines the results of the four following functions: getDBTargets(), getDBTransporters(), getDBCarriers(), and getDBEnzymes().

3.1.2. UniProt

The Multipath package uses the R interface UniProt.ws [6] to query the UniProt database. UniProt.ws contains functions for retrieving, processing and repackaging the UniProt web services. We use the package to get information on proteins, their interactions, and their connections to DrugBank. The information extracted from UniProt is used to add a Protein Layer to a mully graph, with the interactions between the proteins, and the connections to the drugs on the Drug Layer, if contained in the graph. All functions related to UniProt require the UniProt.ws object obtained by calling UniProt.ws() (Figure 3).

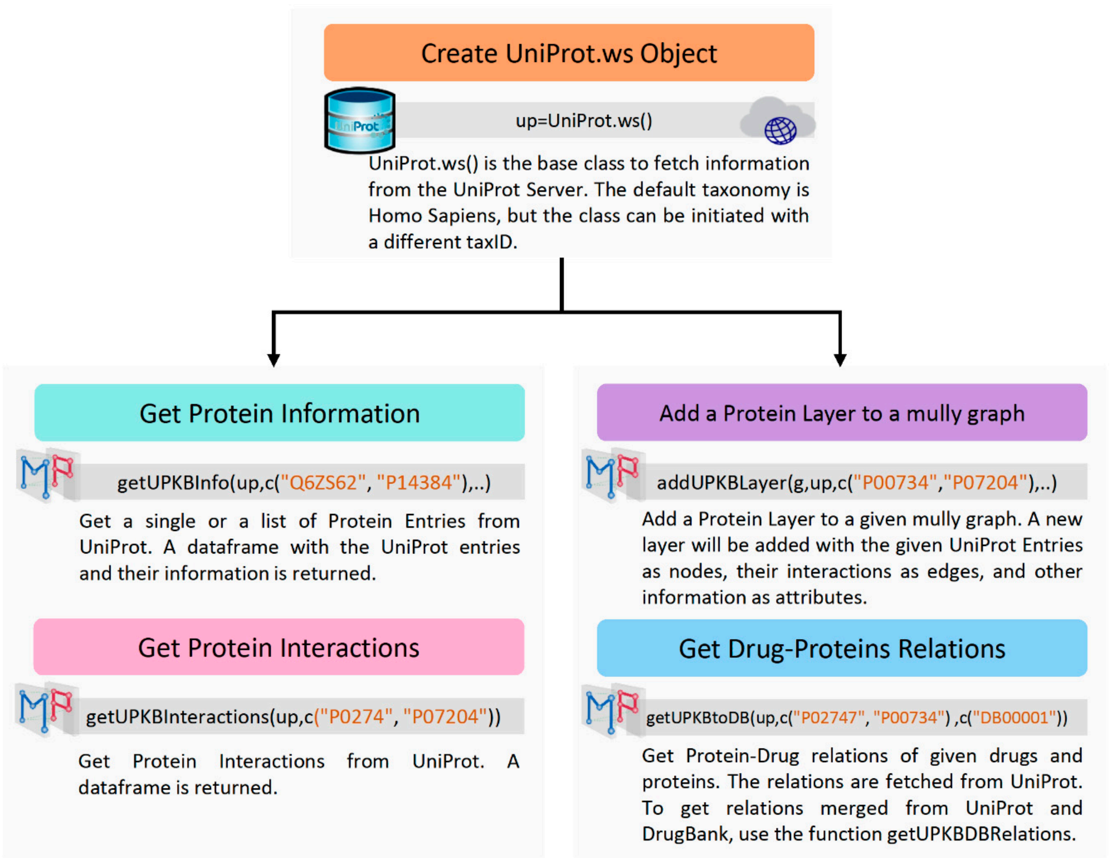

Create UniProt.ws Object
up=UniProt.ws()
UniProt.ws() is the base class to fetch information from the UniProt Server. The default taxonomy is Homo Sapiens, but the class can be initiated with a different taxID.
Get Protein Information
getUPKBInfo(up,c("Q6ZS62", "P14384"),..)
Get a single or a list of Protein Entries from UniProt. A dataframe with the UniProt entries and their information is returned.
Get Protein Interactions
getUPKBInteractions(up,c("P0274", "P07204"))
Get Protein Interactions from UniProt. A dataframe is returned.
Add a Protein Layer to a mully graph
addUPKBLayer(g,up,c("P00734","P07204"),..)
Add a Protein Layer to a given mully graph. A new layer will be added with the given UniProt Entries as nodes, their interactions as edges, and other information as attributes.
Get Drug-Proteins Relations
getUPKBtoDB(up,c("P02747", "P00734") ,c("DB00001"))
Get Protein-Drug relations of given drugs and proteins. The relations are fetched from UniProt. To get relations merged from UniProt and DrugBank, use the function getUPKBDBRelations.

Figure 3. UniProt Workflow Diagram. The Diagram shows the steps to fetch information on Proteins from UniProt using Multipath and the R Package UniProt.ws.

To add the protein layer, the information on a list of proteins can be extracted using addUPKBInfo(), with the UniProt.ws object, the list of UniProt IDs and the columns' names representing the attributes as an input. To get the list of possible columns, the UniProt.ws function columns() can be called. To get the edges between the protein nodes, the function getUPKBInteractions() can be called, returning the list of edges, needed alongside the list of nodes to build the protein layer. This task can be accomplished using the function addUPKBLayer(), with the mully graph, the UniProt.ws object, the list of UniProt IDs, and the list of attributes as arguments.

The connections to the Drug Layer can be obtained by calling the function getUPKBtoDB().

The addition of the Protein and Drug Layers can also be automatically achieved by calling the function multipath(), with the following arguments: the name of the returned mully graph, the UniProt.ws object, the list of UniProt IDs, the DrugBank parsed object, and the list of DrugBank IDs.

3.2. *Pathway Models' Reproducibility*

The main feature of Multipath is the possibility to track modifications applied to the multilayered graph built from the pathway BioPAX file. The user can apply a set of standard procedures, i.e., the addition or removal of nodes, edges, and layers. Every modification is stored in a pathwayView object, which can be used later to undo these modifications, retrace the transformation of the original graph, apply further modifications or compare to other views generated from the same pathway.

4. Discussion

4.1. *Wnt Multipath Model*

The Wnt signaling pathway is an ancient and evolutionarily conserved pathway that regulates crucial aspects of cell fate determination, cell migration, cell polarity, neural patterning, and organogenesis during embryonic development [17]. We generated a multilayered model (Figure 4) of the Signaling by Wnt pathway (BioPAX level 3) [18] from the Reactome Database [9].

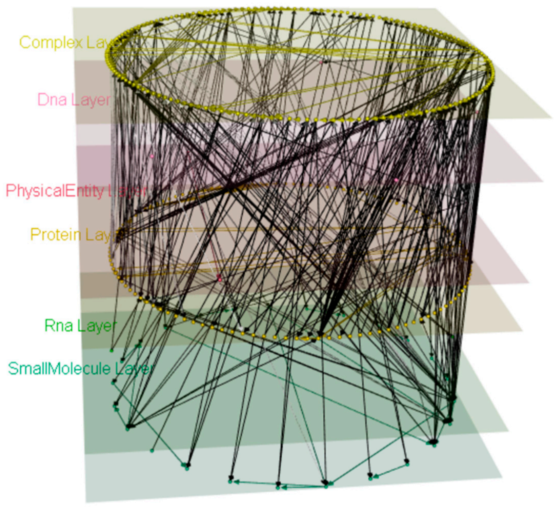


Figure 4. Signaling by Wnt mully model. The model was built using the BioPAX level 3 Signaling by Wnt from Reactome. The model has 6 layers, 311 nodes, and 539 edges.

To build the model and generate the views, we used the R Script in Figure 5.

```
downloadPathway("R-HSA-195721",biopax=3)
wntBiopax=readBiopax("R-HSA-195721.owl")
pathwayID=downloadPathway(wntBiopax,"R-HSA-195721.owl")
wntmully=pathway2Mully(wntBiopax,pathwayID)
plot3d(wntmully,layers=T,vertex.label=NA,edge.width=5,edge.arrow.size=5)

view1=pathwayView(wntmully,"View1")
view1=addStep(view1,action = "remove",element="layer",name="Rna",trans = T)
suppressWarnings(plot3d(view1$modified))

view2=pathwayView(wntmully,"View2")
view2=addStep(view2,action = "remove",element="layer",name="PhysicalEntity",trans=T)
suppressWarnings(plot3d(view2$modified))

view3=pathwayView(wntmully,"View3")
view3=addStep(view3,action = "remove",element="layer",name="Complex",trans=T)
suppressWarnings(plot3d(view3$modified))
```

Figure 5. The script to generate and plot the model and the views. The function pathway2Mully() transforms the pathways parsed in the Biopax object into mully graphs. To plot the graph, we used the function plot3d() from the mully package, and finally, we generated the views using pathwayView() and modified them using the functions addStep() which applies a modification step to the graph.

The downloaded pathway file (4.19 MB) was parsed in 24.56 s and occupies ~5.05 MB of memory space. The transformation to a mully graph was conducted in 19.23 s, and the resulted graph's size is 121.744 KB.

We applied different changes to the model and generated three different views from the model (Figure 6), by deleting the RNA (Figure 6a), Physical Entity (Figure 6b), and Complex (Figure 6c) layers respectively. Upon deletion of the layers, transitive edges derived from the deleted nodes were added. All deleted information can be reversed from the view. The original model (Figure 4) can be reproduced from either of these three views using the function undo.

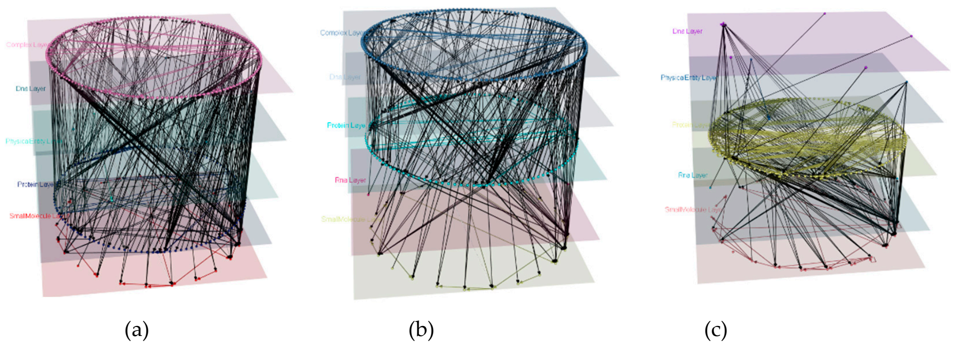


Figure 6. Views generated from the Signaling by Wnt mully model. (**a**) View1 generated by deleting the RNA layer. The graph has 5 layers, 309 nodes, and 533 edges. (**b**) View2 generated by deleting the PhysicalEntity layer. The graph has 5 layers, 308 nodes, and 522 edges (521 edges from the original graph and 1 transitive). (**c**) View3 generated by deleting the Complex layer. The graph has 5 layers, 153 nodes and 800 edges (93 edges from the original graph and 707 transitive).

4.2. Adding a Drug Layer

After creating the Signaling by Wnt Pathway model, we retrieved the Drugs targeting the proteins on the protein layer. We extracted the mappings of the internal IDs to UniProt using the function getExternalIDs(). We obtained the list of Drugs targeting these proteins using getUPKBtoDB(). These IDs were used to add a Drug layer to our model using addDBLayer(), which automatically adds the Drugs and their interactions to the graph. To get the merged Drug–Protein targets from both databases, we used the function getUPKBDBRelations() using the list of external Protein IDs and the list of DrugBank IDs. The edges were added then one by one (Figure 7).

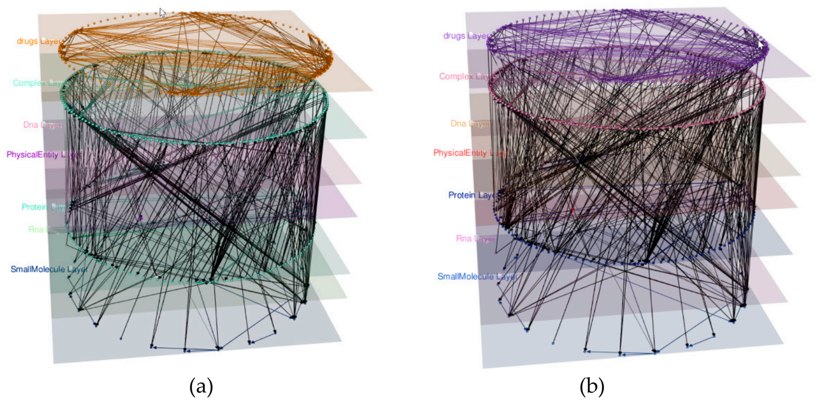


Figure 7. The Signaling by Wnt model with the added Drug Layer. (**a**) The model after adding the Drug layer. The layer contains 83 drugs and 260 Drug Interactions. (**b**) The model after adding the connections between the drugs and proteins' nodes, merged from DrugBank and UniProt. The Drug targets obtained are 97, many of them added as multi-edges, since a single internal protein ID was mapped to multiple UniProt entries.

To retrieve drug information from DrugBank, parsing the downloaded data from DrugBank is required, which is a relatively slow process, fortunately needed only once before calling any drug-related functions. The parsed data is a large list occupying ~553MB of memory space. However, extracting the drug information and interactions, as well as their targets is fast. On the other hand, the retrieval of protein information is accomplished by means of the UniProt.ws R Package. This process does not require any memory space, but we faced a very slow response from the UniProt server multiple times, with no detection of response speed patterns, i.e. specific time of the day, or the amount of information to be retrieved. The web service also recommends querying using a maximum number of 50 UniProt entries' IDs in each single query. Multipath could be updated in the future to support a better API/Tool for this process.

5. Conclusions

Multipath is an R package to generate multilayered models from BioPAX encoded pathway knowledge. The models are modifiable, and all modifications are tracked to allow the reproduction of the graphs. Multipath can also be used to query influencing knowledge databases such as UniProt and DrugBank and integrate them into a multilayered graph. In this paper, we presented the different features of Multipath. We described how to use our package to generate multilayered models from BioPAX files, and integrate different pathway knowledge. We used the Signaling by Wnt pathway as a demonstration, and displayed three different views derived from it. Multipath is open source, and can be downloaded from our GitHub Repository [19]. The reference manual and the vignette of the package were added as Supplementary Materials, respectively as Files S1 and S2.

Supplementary Materials: The following are available online at www.mdpi.com/xxx/s1, File S1: The package's vignette in the PDF format, File S2: The package's Reference Manual.

Author Contributions: Conceptualization, Z.H. and F.K.; methodology, Z.H.; software, Z.H.; validation, Z.H.; investigation, Z.H.; writing—original draft preparation, Z.H.; writing—review and editing, F.K.; visualization, Z.H.; supervision, F.K.; project administration, F.K.; funding acquisition, F.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work is a part of the Multipath and coNfirm projects funded by the GERMAN MINISTRY OF EDUCATION AND RESEARCH (Bundensministerium für Bilding und Forschung), grant FKZ01ZX1508 and FKZ01ZX1708D respectively.

Acknowledgments: I would like to acknowledge the help of my colleagues Florian Auer and Alicia D'Angelo.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Alberts, B.; Johnson, A.; Lewis, J.; Raff, M.; Roberts, K.; Walter, P. *Molecular Biology of the Cell*, 4th ed.; Garland Science: New York, NY, USA, 2002.
2. Goldberg, R.N.; Cary, M.; Demir, E. BioPAX—A community standard for pathway data sharing. *Nat. Biotechnol.* **2010**, *28*, 935–942.
3. Kramer, F.; Bayerlová, M.; Klemm, F.; Bleckmann, A.; Beissbarth, T. rBiopaxParser–an R package to parse, modify and visualize BioPAX data. *Bioinformatics* **2013**, *29*, 520–522. [CrossRef] [PubMed]
4. Hammoud, Z.; Kramer, F. mully: An R Package to Create, Modify and Visualize Multilayered Graphs. *Genes (Basel)* **2018**, *9*, 519. [CrossRef] [PubMed]
5. The Dbparser R Package. Available online: https://github.com/ropensci/dbparser (accessed on 29 September 2020).
6. The UniProt.ws R Interface. Available online: https://bioconductor.org/packages/UniProt.ws/ (accessed on 29 September 2020).
7. Wishart, D.S.; Knox, C.; Guo, A.C.; Shrivastava, S.; Hassanali, M.; Stothard, P.; Chang, Z.; Woolsey, J. DrugBank: A comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Res.* **2006**, *34*, D668–D672. [CrossRef] [PubMed]
8. UniProt: The universal protein knowledgebase. *Nucleic Acids Res.* **2017**, *45*, D158–D169. [CrossRef] [PubMed]
9. Croft, D.; Mundo, A.F.; Haw, R.; Milacic, M.; Weiser, J.; Wu, G.; Caudy, M.; Garapati, P.; Gillespie, M.; Kamdar, M.R.; et al. The Reactome pathway knowledgebase. *Nucleic Acids Res.* **2014**, *42*, D472–D477. [CrossRef] [PubMed]
10. Bader, G.D.; Cary, M.P.; Sander, C. Pathguide: A Pathway Resource List. *Nucleic Acids Res.* **2006**, *34* (Suppl. 1), D504–D506. [CrossRef] [PubMed]

11. Degtyarenko, K.; De Matos, P.; Ennis, M.; Hastings, J.; Zbinden, M.; McNaught, A.; Alcántara, R.; Darsow, M.; Guedj, M.; Ashburner, M. ChEBI: A database and ontology for chemical entities of biological interest. *Nucleic Acids Res.* **2008**, *36*, D344–D350. [CrossRef] [PubMed]
12. Kim, S.; Thiessen, P.A.; Bolton, E.E.; Chen, J.; Fu, G.; Gindulyte, A.; Han, L.; He, J.; He, S.; Shoemaker, B.A.; et al. PubChem Substance and Compound databases. *Nucleic Acids Res.* **2016**, *44*, D1202–D1213. [CrossRef] [PubMed]
13. Szöcs, E.; Stirling, T.; Scott, E.R.; Scharmüller, A.; Schäfer, R.B. webchem: An R Package to Retrieve Chemical Information from the Web. *J. Stat. Softw.* **2020**, *93*, 1–17. [CrossRef]
14. Kanehisa, M.; Goto, S. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res.* **2000**, *28*, 27–30. [CrossRef] [PubMed]
15. Hubbard, T.; Barker, D.; Birney, E.; Cameron, G.; Chen, Y.; Clark, L.; Cox, T.; Cuff, J.; Curwen, V.; Down, T.; et al. The Ensembl genome database project. *Nucleic Acids Res.* **2002**, *30*, 38–41. [CrossRef] [PubMed]
16. The DrugBank Website. Available online: drugbank.ca (accessed on 29 September 2020).
17. Komiya, Y.; Habas, R. Wnt signal transduction pathways. *Organogenesis* **2008**, *4*, 68–75. [CrossRef] [PubMed]
18. Signaling by Wnt. Available online: https://reactome.org/content/detail/R-HSA-195721 (accessed on 29 September 2020).
19. The Multipath R Package. Available online: https://github.com/frankkramer-lab/Multipath (accessed on 29 September 2020).

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

Multipath Paper Supplementary Materials

Vignette

Zaynab Hammoud

Multipath - an R package to create integrated reproducible multilayered pathway models

true

2020-11-04

Introduction

Biological pathway data integration has become a topic of interest in the past years. This interest originates essentially from the continuously increasing size of existing prior knowledge as well as from the many challenges scientists face when studying biological pathways. Multipath is a framework that aims at helping re-trace the use of specific pathway knowledge in specific publications, and easing the data integration of multiple pathway types and further influencing knowledge sources. Using Multipath, BioPax-encoded pathways can be parsed and embedded into multilayered graphs. Modifications can be applied to these graphs to generate different views. The package is implemented as a part of the Multipath Project directed by Dr. Frank Kramer .

Installation

Preinstallation

Multipath depends on multiple packages. The packages are the following: UniProt.ws, dbparser, rBiopaxParser, mully, TCGAretriever, stringr, stringi, svMisc, uuid, dplyr, crayon Please make sure to install the packages UniProt.ws,rBiopaxParser and mully before using the package.

To install the UniProt.ws package:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("UniProt.ws")
```

To install the mully package:

```
require(devtools)
install_github("frankkramer-lab/mully")
library(mully)
```

To install the rBiopaxParser package:

```
require(devtools)
install_github("frankkramer-lab/rBiopaxParser")
library(rBiopaxParser)
```

Installation via Github

```
require(devtools)
install_github("frankkramer-lab/Multipath")
library(Multipath)
```

Available Functions

addDBLayer

Add a drug layer to a mully graph

This function is used to add a DrugBank layer to an existing mully model. The function needs the following arguments:

- **g** - The mully graph
- **drugList** - The list of DrugBank Ids of the drugs to be added. This argument can be either a string (one drug) or a list of strings (multiple drugs)
- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`

The function returns a mully graph with the added drug layer

Example

```
g=mully("DrugBank",direct=T)
data=loadDBXML("DrugBank.xml")
g=addDBLayer(g,data,c("DB00001","DB06605"))
```

addStep

Track a modification of a graph

The function saves a modification applied to a mully graph. It applies the step to the graph and saves the modification step in the pathwayView Object. Not all of the arguments are mandatory, they depend on the step that has to be applied. The function needs the following arguments:

- **v** - The input view in which the modification should be saved
- **action** - The type of action to be applied. Can either be “add” or "remove
- **element** - The type of the element to be modified. Can either be “node”, “edge”, or “layer”
- **name** - The name of the element to be modified. This argument is only mandatory for nodes and edges
- **layername** - The layer name. This argument is only mandatory for action “add” and element “node”
- **V1** - The start node of an edge. This argument is only mandatory for element “edge”
- **V2** - The end node of an edge. This argument is only mandatory for element “edge”
- **attributes** - The named list of attributes of the element. This argument is required only for action “add”. It is optional for both elements “node” and “edge”, but mandatory if the edge alread exists
- **multi** - A boolean whether to select multi-edges or not. This is only mandatory for action “remove” and element “edge”. By default set to FALSE, in which case the attributes of the specified edge should be given

- **trans** - A boolean whether to add transitive edges upon removal of nodes or layers

The function returns the view with the added step. ***Example***

```r
g=mully:::demo()
view=pathwayView(g,"View1")
view=addStep(view,"remove","layer","disease")
```

addUPKBLayer

Add a protein layer to a mully graph

This function is used to add a UniProt protein layer to an existing mully model. The function needs the following arguments: - **g** - The mully graph - **up** - The UniProt.ws Object - **proteinList** - The list of UniProt Ids of the proteins to be added - **col** - The list of attributes associated to the UniProtKB Entries to be retrieved

The function returns the mully graph with the added UniProt layer The function should be preceded by `UniProt.ws()` to get the UniProt.ws Object

Example

```r
up=UniProt.ws()
g=mully("UniProt")
g=addUPKBLayer(g,up,proteinList=c("P02747","P00734","P07204"),col=c("UNIPROTKB","PROTEIN-NAMES"))
```

downloadPathway

Download Reactome Pathways in BioPAX level 2 and 3

This function is used to download one or a list of pathways, encoded in BioPAX level 2 or 3. The function needs the following arguments:

- **pathwayID** - The Reactome ID or list of IDs of the pathways to be downloaded. The ID should start with R-HSA-.
- **biopaxLevel** - The BioPAX Level, 2 or 3. By default set to 3.
- **destDirectory** - The Directory in which the Pathway Files should be saved. If missing, the files are saved in the working directory. The Reactome IDs are used to name the files.
- **overwrite** - A Boolean whether to overwrite existing files with the same name.

The function returns the path to the directory in which the files are downloaded.

Example

```r
downloadPathway(c("R-HSA-195721","R-HSA-9609507"),biopaxLevel=3,overwrite=T)
```

getAllUPKB

Get all proteins' entries from UniProt

The function is used to fetch all protein entries from UniProt. The function needs the following arguments: - **up** - The UniProt.ws Object

The function returns a dataframe containing the Protein's entries with the ID and Name.

Should be preceded by `UniProt.ws()` to get the UniProt.ws Object

Example

```
up=UniProt.ws()
allProteins=getAllUPKB(up)
```

getDBCarriers

Get the Carriers Protein Targets of given DrugBank drugs

Protein Targeted by Drugs are divided in DrugBank into 4 types: Targets, Enzymes, Carriers and Transporters. This function is used to extract the carriers from the dataframe containing the information on the drugs parsed from the DrugBank XML File.

The function needs the following arguments:

- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drugList** - The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

The function returns a dataframe containing all information on the carriers targeted by the given drug list.

Example

```
data=loadDBXML(DrugBankFilePath)
getDBCarriers(data,"DB00001")
```

getDBDrug

Get DrugBank drug entry

This function extracts infromation on one or a list of Drugs from the dataframe parsed from the DrugBank XML file. The function needs the following arguments:

- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drug** - The ID or list of IDs of the DrugBank drug entries starting with "DB"

This function returns a dataframe containing the DrugBank entry with its information

Example

```
data=loadDBXML(DrugBankFilePath)
getDBDrug(data,"DB00001")
```

getDBDrugInteractions

Get DrugBank Drug to Drug Interactions This function is used to extract Drug Interactions from the dataframe containing the information on the Drug in DrugBank, parsed from the downloaded XML File.

The function needs the following arguments:

- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drug** - The ID of the DrugBank drug entry starting with “DB”. This argument can be either a string (one drug) or a list of strings (multiple drugs).

The function returns a dataframe containing the DrugBank interactions in which the given drug is involved
Example

```
data=loadDBXML("DrugBank.xml")
getDBDrugInteractions(data,"DB06605")
```

getDBEnzymes

Get the Enzyme Protein Targets of given DrugBank drugs

Protein Targeted by Drugs are divided in DrugBank into 4 types: Targets, Enzymes, Carriers and Transporters. This function is used to extract the enzymes from the dataframe containing the information on the drugs parsed from the DrugBank XML File.

The function needs the following arguments:

- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drugList** - The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

The function returns a dataframe containing all information on the enzymes targeted by the given drug list.

Example

```
data=loadDBXML(DrugBankFilePath)
getDBEnzymes(data,"DB00001")
```

getDBTargets

Get the Target Protein Targets of given DrugBank drugs

Protein Targeted by Drugs are divided in DrugBank into 4 types: Targets, Enzymes, Carriers and Transporters. This function is used to extract the targets from the dataframe containing the information on the drugs parsed from the DrugBank XML File.

The function needs the following arguments:

- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drugList** - The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

The function returns a dataframe containing all information on the targets of the given drug list.

Example

```
data=loadDBXML(DrugBankFilePath)
getDBTargets(data,"DB00001")
```

getDBTransporters

Get the Transporters Protein Targets of given DrugBank drugs

Protein Targeted by Drugs are divided in DrugBank into 4 types: Targets, Enzymes, Carriers and Transporters. This function is used to extract the transporters from the dataframe containing the information on the drugs parsed from the DrugBank XML File.

The function needs the following arguments:

- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drugList** - The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

The function returns a dataframe containing all information on the transporters targeted by the given drug list.

Example

```
data=loadDBXML(DrugBankFilePath)
getDBTransporters(data,"DB00001")
```

getDBtoUPKB

Get DrugBank Drugs to UniProt Proteins Relations from DrugBank

This function is used to extract Drug Targets from the dataframe containing the information on the drugs parsed from the DrugBank XML File. It merges the targets returned by 4 functions: enzymes, targets, transporters and carriers. The function needs the following arguments:

- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drugList** - The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)
- **proteinList** - The list of UniProt Ids of the proteins

The function returns a dataframe containing the connections between DrugBank drugs and UniProt proteins retrieved from DrugBank.

Example

```
data=loadDBXML("DrugBank.xml")
getDBtoUPKB(data,c("DB00001","DB00002","DB00006"),c("P02747","P00734","P07204","P05164"))
```

getPathwayID

Get internal pathway ID in a BioPAX file

This function is used to get the internal ID of a pathway in a parsed BioPAX object. A BioPAX file can contain multiple pathways, indexed internally using ID starting with "Pathway" followed by the number of the pathway. Each pathway in the file has a Reactome and an internal ID. The latter can be extracted using this function. This should be preceded by `readBiopax(filepath)` to obtain the biopax object The function needs the following arguments:

- **biopax** - The biopax object
- **reactomeID** - The Reactome ID of the pathway

The function returns the internal ID of the pathway in the parsed BioPAX object.

Example

```
biopax=readBiopax("pi3k.owl")
id=getPathwayID(biopax,"R-HSA-167057")
pi3kmully=pathway2mully(biopax,id)
```

getUPKBDBRelations

Get Protein and Drugs relations from UniProt and DrugBank

The function is used to obtain drug targets from UniProt and DrugBank. It combines the returned relations from both functions `getDBtoUPKB` and `getUPKBtoDB`. The function needs the following arguments:

- **up** - The UniProt.ws Object
- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)` -**proteinList** - The list of UniProt Ids of the proteins
- **drugList** - The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

The function returns a dataframe containing the connections between DrugBank drugs and UniProt proteins retrieved from DrugBank and UniProt. The function should be preceded by:

1. `UniProt.ws()` to get the UniProt.ws Object
2. `loadDBXML(DrugBankFile)` to get the argument data

Example

```
up=UniProt.ws()
data=loadDBXML("DrugBank.xml")
relations=getUPKBDBRelations(up,data,proteinList=c("P02747","P00734","P07204","P05164"),drugList=c("DB0
```

getUPKBInfo

Get Proteins from UniProtKB

The function is used to fetch information on a list of protein entries from UniProt. The function needs the following arguments: - **up** - The UniProt.ws Object - **proteins** - The list of UniProtKB Proteins ID to be retrieved - **col** - The list of attributes associated to the UniProtKB Entries to be retrieved

The function returns a dataframe containing the protein entries with the selected attributes. To get the list of possible columns, you can call `columns(UniProt.ws())`. The function should be preceded by `UniProt.ws()` to get the UniProt.ws Object.

Example

```
up <- UniProt.ws()
getUPKBInfo(up,c("Q6ZS62","P14384","P40259"),c("PROTEIN-NAMES","DRUGBANK","GO","REACTOME"))
```

getUPKBInteractions

Get the interactions of given proteins from UniProt

The function is used to fetch interactions between proteins from the UniProt Database. The function needs the following arguments:

- **up** - The UniProt.ws Object
- **proteins** - The list of proteins of which the interactions should be retrieved

The function returns a dataframe containing the interactions between the given proteins. The function should be preceded by `UniProt.ws()` to get the UniProt.ws Object.

Example

```
up=UniProt.ws()
interactions=getUPKBInteractions(up,c("P02747","P07204","P00734"))
```

getUPKBtoDB

Get UniProt Proteins to DrugBank Drugs relations from UniProt

This function is used to fetch relations between a list of proteins and a list of drugs from the UniProt Database. The function needs the following arguments:

- **up** - The UniProt.ws Object
- **proteinList** - The list of UniProt Ids of the proteins
- **drugList** - The ID of the DrugBank drug entry starting with "DB". This argument can be either a string (one drug) or a list of strings (multiple drugs).

The function returns a dataframe containing the connections between UniProt proteins and DrugBank drugs retrieved from UniProt. The function should be preceded by `UniProt.ws()` to get the UniProt.ws Object.

Example

```
up=UniProt.ws()
getUPKBtoDB(up,c("P02747","P00734","P07204"),c("DB00001","DB00002"))
```

loadDBXML

Load DrugBank XML file

This function is used to read and parse the file downloaded from the DrugBank Database containing the complete information on the drug entries. The function needs the following argument:

- **file** - The path to the DrugBank XML file. This can be downloaded from the DrugBank official Website (drugbank.ca). An account with an institutional e-mail is required.

This function returns a dataframe containing the parsed information from DrugBank. This can be used to extract any additional information on the DrugBank entries

This function should be called before using any function to query the DrugBank database. Since the parsing of DrugBank takes time, this function should only be called once.

Example

```
data=loadDBXML("DrugBank.xml")
```

multipath

Generate Multipath Graph from General Data

This function is used to generate a mully graph from a list of drugs and proteins. The function creates a multilayered graph with a drug and protein layer, and adds the inter- and intractions to it. The function needs the following arguments:

- **name** - The name of the graph to be generated
- **up** - The `Uniprot.ws()` object
- **proteinList** - The list of proteins of which the interactions should be retrieved
- **data** - The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function `loadDBXML(DrugBankFile)`
- **drugList** - The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

The function returns a mully graph with the added data. The function should be preceded by:

1. `UniProt.ws()` to get the UniProt.ws Object
2. `loadDBXML(DrugBankFile)` to get the argument data

Example

```
up=UniProt.ws()
data=loadDBXML("DrugBank.xml")
g=multipath(name="MyMultipath",up=up,proteinList=c("P02747","P00734","P07204","P05164"),data=data,drugL
```

pathway2Mully

Build a mully graph from a given pathway This function builds a multilayered mully graph of a BioPAX encoded pathway. To run this function, the user needs to parse the file. It should be preceded by `readBiopax(filepath)` to obtain the biopax object. The function needs the following arguments:

- **biopax** - The BioPaX object containing the parsed data from an OWL file. This can be obtained using `readBiopax(filepath)`
- **pathwayID** - The internal ID of the pathway in the biopax object. To obtain the internal ID, the function `getPathwayID(biopax,reactomeID)` can be called

The function returns a mully graph built from the given pathway.

Example

```
biopax=readBiopax("pi3k.owl")
pi3kmully=pathway2mully(biopax,"pathway1")
```

pathwayView

Create an empty view

The function is used to create a pathwayView in order to track the modifications applied to a mully graph. The ocject pathwayView contains different information on the View, including the timestamp of creation and last modification, the original and final version of the graph, and the dataframe containing the modification steps. The function needs the following arguments:

- **g** - The input graph
- **name** - The name of the view

The function returns an empty pathwayView Object.

Example

```
view=pathwayView(mully("myMully",T),"View1")
```

print,pathwayView

Print Function

The function is used to print the pathwayView Object. The function needs the following arguments:

- **v** - The input pathwayView to be printed

undo

Undo a modification step in a view

The function reverses changes applied to a mully graph, saved in a pathwayView Object. The function needs the following arguments:

- **v** - The input view
- **stps** - The number of steps to undo. This number referes to the number of unique steps' IDs to be removed, i.e. entries of steps in the view with similar stepID count as 1

The function returns The view with the undone modifications. ## wntpathway **Demo function for Wnt Pathway Views** The function is a demo function that create a pathway mully graph from a BioPAX encoded file of the Signaling by Wnt Pathway. The function reads and parses the file, creates the mully graph, and generates 3 different views from the graph by deleting the RNA, Complex, and Physical Entity Layers. The function needs the following arguments:

- **file** - The link to the Wnt Pathway bioPAX file

Example

```
wntpathway("wnt_reactome.owl")
```

Reference Manual

Package ‘Multipath’

December 10, 2020

Type Package

Title Pathway Data Integration With Influencing Knowledge Using Multi-Layered Graphs

Version 1.0.3

Author Zaynab Hammoud

Maintainer Zaynab Hammoud <zaynabhassanhammoud@gmail.com>

Description Allows the integration of pathway data with influencing knowledge from different databases, like DrugBank, Reactome and UniProt, by using multilayered graphs.

License GPL (¿= 2)

Encoding UTF-8

LazyData true

Depends stringr

RoxygenNote 7.1.1

Suggests knitr,
rmarkdown

Imports mully (¿= 2.1.26),
UniProt.ws,
dbparser,
rBiopaxParser,
TCGAretriever,
svMisc,
uuid,
dplyr,
crayon,
igraph,
RCurl,
graph

VignetteBuilder knitr

R topics documented:

`addDBLayer` *Add a drug layer to a mully graph*

Description

Add a drug layer to a mully graph

Usage

```
addDBLayer(g, data, drugList)
```

Arguments

`g`	The mully graph
`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drugList`	The list of DrugBank Ids of the drugs to be added. This argument can be either a string (one drug) or a list of strings (multiple drugs)

Value

A mully graph with the added drug layer

Examples

```
## Not run:
data=readDBXML(DBXMLFilePath)
g=mully("DrugBank",direct=T)
g=addDBLayer(g,data,c("DB00001","DB06605"))

## End(Not run)
```

addStep — *Track a modification of a graph*

Description

Track a modification of a graph

Usage

```
addStep(
  v,
  action,
  element,
  name = NA,
  layername = NA,
  V1 = NA,
  V2 = NA,
  attributes = NA,
  multi = F,
  trans = T
)
```

Arguments

`v`	The input view in which the modification should be saved
`action`	The type of action to be applied. Can either be "add" or "remove
`element`	The type of the element to be modified. Can either be "node", "edge", or "layer"
`name`	The name of the element to be modified. This argument is only mandatory for nodes and edges
`layername`	The layer name. This argument is only mandatory for action "add" and element "node"
`V1`	The start node of an edge. This argument is only mandatory for element "edge"
`V2`	The end node of an edge. This argument is only mandatory for element "edge"
`attributes`	The named list of attributes of the element. This argument is required only for action "add". It is optional for both elements "node" and "edge", but mandatory if the edge alread exists
`multi`	A boolean whether to select multi-edges or not. This is only mandatory for action "remove" and element "edge". By default set to FALSE, in which case the attributes of the specified edge should be given
`trans`	A boolean whether to add transitive edges upon removal of nodes or layers

Value

The View with the added step

Examples

```
## Not run:
g=mully:::demo()
view=pathwayView(g,"View1")
view=addStep(view,"remove","layer","")

## End(Not run)
```

---

addUPKBLayer *Add a protein layer to a mully graph*

Description

Add a protein layer to a mully graph

Usage

```
addUPKBLayer(
  g,
  up,
  proteinList,
  col = c("UNIPROTKB", "PROTEIN-NAMES", "ORGANISM")
)
```

Arguments

`g`	The mully graph
`up`	The UniProt.ws Object
`proteinList`	The list of UniProt Ids of the proteins to be added
`col`	The list of attributes associated to the UniProtKB Entries to be retrieved

Value

The mully graph with the added UniProt layer

Note

Should be preceded by UniProt.ws() to get the UniProt.ws Object

Examples

```
## Not run:
up=UniProt.ws()
g=mully("UniProt")
g=addUPKBLayer(g,up,proteinList=c("P02747","P00734","P07204"),col=c("UNIPROTKB","PROTEIN-NAMES"))

## End(Not run)
```

`downloadPathway` *Download Reactome Pathways in BioPAX level 2 and 3*

Description

Download Reactome Pathways in BioPAX level 2 and 3

Usage

```
downloadPathway(pathwayID, biopaxLevel = "3", destDirectory, overwrite = F)
```

Arguments

`pathwayID`	The Reactome ID or list of IDs of the pathways to be downloaded. The ID should start with R-HSA-.
`biopaxLevel`	The BioPAX Level, 2 or 3. By default set to 3.
`destDirectory`	The Directory in which the Pathway Files should be saved. If missing, the files are saved in the working directory. The Reactome IDs are used to name the files.
`overwrite`	A Boolean whether to overwrite existing files with the same name.

Value

The Directory in which the files are saved.

Examples

```
## Not run:
downloadPathway(c("R-HSA-195721","R-HSA-9609507"),biopaxLevel=3,overwrite=T)

## End(Not run)
```

`getAllUPKB` *Get all proteins' entries from UniProt*

Description

Get all proteins' entries from UniProt

Usage

```
getAllUPKB(up)
```

Arguments

`up`	The UniProt.ws Object

Value

a dataframe containing the Protein's entries with the ID and Name

Note

Should be preceded by UniProt.ws() to get the UniProt.ws Object

Examples

```
## Not run:
up=UniProt.ws()
allProteins=getAllUPKB(up)

## End(Not run)
```

---

getDBCarriers *Get the carrier proteins involved in movement of given drugs across biological membranes*

Description

Get the carrier proteins involved in movement of given drugs across biological membranes

Usage

```
getDBCarriers(data, drugList)
```

Arguments

`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drugList`	The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

Value

A dataframe containing all information on the carrier proteins involved in movement of the given drugs across biological membranes

---

getDBDrug *Get DrugBank drug entry*

Description

Get DrugBank drug entry

Usage

```
getDBDrug(data, drug)
```

Arguments

`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drug`	The ID or list of IDs of the DrugBank drug entries starting with ”DB”

Value

A dataframe containing the DrugBank entry with its information

Examples

```
## Not run:
data=loadDBXML(DrugBankFilePath)
getDBDrug(data, "DB00001")

## End(Not run)
```

`getDBDrugInteractions` *Get DrugBank Drug to Drug Interactions*

Description

Get DrugBank Drug to Drug Interactions

Usage

```
getDBDrugInteractions(data, drug)
```

Arguments

`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drug`	The ID of the DrugBank drug entry starting with "DB". This argument can be either a string (one drug) or a list of strings (multiple drugs).

Value

A dataframe containing the DrugBank interactions in which the given drug is involved

Examples

```
## Not run:
data=loadDBXML(DBXMLFilePath)
getDBDrugInteractions(data,"DB06605")

## End(Not run)
```

getDBEnzymes — *Get the enzymes inhibited/induced or involved in metabolism by given DrugBank drugs*

Description

Get the enzymes inhibited/induced or involved in metabolism by given DrugBank drugs

Usage

```
getDBEnzymes(data, drugList)
```

Arguments

`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drugList`	The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

Value

A dataframe containing all information on the enzymes inhibited/induced or involved in metabolism by the given drug list

getDBTargets — *Get the targets of given DrugBank drugs*

Description

Get the targets of given DrugBank drugs

Usage

```
getDBTargets(data, drugList)
```

Arguments

`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drugList`	The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

Value

A dataframe containing all information on the targets of the given drug list

getDBtoUPKB — *Get DrugBank Drugs to UniProt Proteins Relations from DrugBank*

Description

Get DrugBank Drugs to UniProt Proteins Relations from DrugBank

Usage

```
getDBtoUPKB(data, drugList, proteinList)
```

Arguments

`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drugList`	The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)
`proteinList`	The list of UniProt Ids of the proteins

Value

A dataframe containing the connections between DrugBank drugs and UniProt proteins retrieved from DrugBank

Examples

```
## Not run:
data=readDBXML(DBXMLFilePath)
getDBtoUPKB(data,c("DB00001","DB00002","DB00006"),c("P02747","P00734","P07204","P05164"))

## End(Not run)
```

getDBTransporters — *Get the transporter proteins involved in movement of given drugs across biological membranes*

Description

Get the transporter proteins involved in movement of given drugs across biological membranes

Usage

```
getDBTransporters(data, drugList)
```

Arguments

`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drugList`	The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

Value

A dataframe containing all information on the transporter proteins involved in movement of the given drugs across biological membranes

---

`getExternalIDs` *Get External Database IDs of nodes*

Description

Get External Database IDs of nodes

Usage

```
getExternalIDs(biopax, nodes, database)
```

Arguments

`biopax`	The biopax object
`nodes`	The list of internal IDs of the nodes
`database`	The name of the database

Value

A dataframe with the mappings between the internal and external IDs

Examples

```
## Not run:
biopax=readBiopax(pi3k.owl)
getExternalIDs(wntBiopax,c("Protein1","Protein2"),"UniProt")

## End(Not run)
```

`getPathwayID` *Get internal pathway ID in a BioPAX file*

Description

Get internal pathway ID in a BioPAX file

Usage

```
getPathwayID(biopax, reactomeID)
```

Arguments

`biopax`	The biopax object
`reactomeID`	The Reactome ID of the pathway

Value

The internal ID of the pathway in the parsed BioPAX object

Note

This should be preceded by readBiopax(filepath) to obtain the biopax object

Examples

```
## Not run:
biopax=readBiopax(pi3k.owl)
id=getPathwayID(biopax,"R-HSA-167057")
pi3kmully=pathway2mully(biopax,id)

## End(Not run)
```

`getUPKBDBRelations` *Get Protein and Drugs relations from UniProt and DrugBank*

Description

Get Protein and Drugs relations from UniProt and DrugBank

Usage

```
getUPKBDBRelations(up, data, proteinList, drugList)
```

Arguments

`up`	The UniProt.ws Object
`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`proteinList`	The list of UniProt Ids of the proteins
`drugList`	The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

Value

A dataframe containing the connections between DrugBank drugs and UniProt proteins retrieved from DrugBank and UniProt

Note

Should be preceded by: 1. UniProt.ws() to get the UniProt.ws Object 2. loadDBXML(DrugBankFile) to get the argument data

Examples

```
## Not run:
up=UniProt.ws()
data=readDBXML(DBXMLFilePath)
relations=getUPKBDBRelations(up,data,c("P02747","P05164"),c("DB00001","DB00006"))

## End(Not run)
```

getUPKBInfo *Get Proteins from UniProtKB*

Description

Get Proteins from UniProtKB

Usage

```
getUPKBInfo(up, proteins, col)
```

Arguments

`up`	The UniProt.ws Object
`proteins`	The list of UniProtKB Proteins ID to be retrieved
`col`	The list of attributes associated to the UniProtKB Entries to be retrieved

Value

a dataframe containing the protein entries with the selected attributes

Note

Should be preceded by UniProt.ws() to get the UniProt.ws Object To get the list of possible columns, you can call columns(UniProt.ws())

Examples

```
## Not run:
up <- UniProt.ws()
getUPKBInfo(up,c("Q6ZS62","P14384"),c("PROTEIN-NAMES","GO"))

## End(Not run)
```

getUPKBInteractions *Get the interactions of given proteins from UniProt*

Description

Get the interactions of given proteins from UniProt

Usage

```
getUPKBInteractions(up, proteins)
```

Arguments

`up`	The UniProt.ws Object
`proteins`	The list of proteins of which the interactions should be retrieved

Value

A dataframe containing the interactions between the given proteins

Note

Should be preceded by UniProt.ws() to get the UniProt.ws Object

Examples

```
## Not run:
up=UniProt.ws()
interactions=getUPKBInteractions(up,c("P02747","P07204","P00734"))

## End(Not run)
```

`getUPKBtoDB` *Get UniProt Proteins to DrugBank Drugs relations from UniProt*

Description

Get UniProt Proteins to DrugBank Drugs relations from UniProt

Usage

```
getUPKBtoDB(up, proteinList, drugList)
```

Arguments

`up`	The UniProt.ws Object
`proteinList`	The list of UniProt Ids of the proteins
`drugList`	The ID of the DrugBank drug entry starting with "DB". This argument can be either a string (one drug) or a list of strings (multiple drugs).

Value

A dataframe containing the connections between UniProt proteins and DrugBank drugs retrieved from UniProt

Note

Should be preceded by UniProt.ws() to get the UniProt.ws Object

Examples

```
## Not run:
up=UniProt.ws()
getUPKBtoDB(up,c("P02747","P00734","P07204"),c("DB00001","DB00002"))

## End(Not run)
```

`loadDBXML` *Load DrugBank XML file*

Description

Load DrugBank XML file

Usage

```
loadDBXML(file)
```

Arguments

`file`	The path to the DrugBank XML file. This can be downloaded from the DrugBank official Website (drugbank.ca). An account with an institutional e-mail is required.

Value

A dataframe containing the parsed information from DrugBank. This can be used to extract any additional information on the DrugBank entries

Note

This function should be called before using any function to query the DrugBank database. Since the parsing of DrugBank takes time, this function should only be called once.

`multipath` *Generate Multipath Graph from General Data*

Description

Generate Multipath Graph from General Data

Usage

```
multipath(
  name = "Multipath",
  up = NA,
  proteinList = NA,
  data = NA,
  drugList = NA
)
```

Arguments

`name`	The name of the graph to be generated
`up`	The Uniprot.ws() object
`proteinList`	The list of proteins of which the interactions should be retrieved
`data`	The dataframe containing the parsed information of DrugBank. This argument can be obtained using the function loadDBXML(DrugBankFile)
`drugList`	The list of DrugBank Ids of the drugs. This argument can be either a string (one drug) or a list of strings (multiple drugs)

Value

A mully graph with the added data

`pathway2Mully` *Build a mully graph from a given pathway*

Description

Build a mully graph from a given pathway

Usage

```
pathway2Mully(biopax, pathwayID)
```

Arguments

`biopax`	The BioPaX object containing the parsed data from an OWL file. This can be obtained using readBiopax(filepath)
`pathwayID`	The ID of the pathway in the biopax object

Value

A mully graph built from the given pathway

Note

This should be preceded by readBiopax(filepath) to obtain the biopax object

Examples

```
## Not run:
biopax=readBiopax(pi3k.owl)
pi3kmully=pathway2mully(biopax,"pathway1")

## End(Not run)
```

pathwayView *Create an empty view*

Description

Create an empty view

Usage

```
pathwayView(g, name)
```

Arguments

g	The input graph
name	The name of the view

Value

An empty view

Examples

```
## Not run:
view=pathwayView(mully("myMully",T),"View1")

## End(Not run)
```

print.pathwayView *Print function*

Description

Print function

Usage

```
## S3 method for class 'pathwayView'
print(x, ...)
```

Arguments

x	The input View to be printed
...	any other parameteres passed to `print`

`undo` *Undo a modification step in a view*

Description

Undo a modification step in a view

Usage

```
undo(v, stps = 1)
```

Arguments

`v`	The input view
`stps`	The number of steps to undo. This number referes to the number of unique steps' IDs to be removed, i.e. entries of steps in the view with similar stepID count as 1

Value

The view with the undone modifications

`wntpathway` *Demo function for Wnt Pathway Views*

Description

Demo function for Wnt Pathway Views

Usage

```
wntpathway(file)
```

Arguments

`file`	The link to the Wnt Pathway biopax file

Index

Appendix D

Publication “Identification of potential therapeutic targets for host-directed Leishmaniasis treatment and repositioning drugs using network-based approaches”

Research Article

Identification of potential therapeutic targets for host-directed Leishmaniasis treatment and repositioning drugs using network-based approaches

J. Eduardo Martinez-Hernandez[1,2,3,4], Zaynab Hammoud[5], Frank Kramer[5], Rubens Monte-Neto[6], Vinicius Maracaja-Coutinho[3,4], Alberto J. M. Martin[4*]

[1] Programa de Doctorado en Genómica Integrativa, Vicerrectoría de Investigación, Universidad Mayor, Santiago, Chile.
[2] Network Biology Laboratory, Centro de Genómica y Bioinformática, Facultad de Ciencias, Universidad Mayor, Santiago, Chile.
[3] Advanced Center for Chronic Diseases - ACCDiS, Facultad de Ciencias Químicas y Farmacéuticas, Universidad de Chile, Santiago, Chile.
[4] Centro de Modelamiento Molecular, Biofísica y Bioinformática – CM2B2, Facultad de Ciencias Químicas y Farmacéuticas, Universidad de Chile, Santiago, Chile.
[5] IT-Infrastructure for Translational Medical Research, University of Augsburg, 86159 Augsburg, Germany
[6] Instituto René Rachou - Fundação Oswaldo Cruz, Belo Horizonte, Minas Gerais, Brazil.

Abstract

Leishmania parasites are the causal agent of several diseases called *Leishmaniasis*. Host-directed therapies present an alternative that promises to be useful for antileishmanial therapy. We aimed to identify potential novel therapeutic target genes for host-directed treatment and to get their target-drug interaction in order to obtain new drugs that can be repositioned for their use as antileishmanial therapies. We employed context-specific gene regulatory networks to model the regulation dynamics of *Leishmania* major infected human macrophages in four time series that allow us to identify several genes as new targets. The pathway mapping and drug-target multilayered analysis reveal the presence of 5 genes that can be proposed as potential new therapeutic targets for host-directed therapies. Also, we identified the potential of repositioning 11 drugs that target our proposed genes. This work constitutes an effort to characterize novel antileishmanial therapies by a combination of network-based approaches, demonstrating that our proposed targets and drugs can be used as a new alternative for *Leishmania* major infection treatment.

1. Introduction

Leishmaniasis is a group of vector-borne neglected tropical diseases caused by *Leishmania* parasites (Burza, Croft, and Boelaert 2018). The clinical manifestations range from self-healing skin ulcerations for *cutaneous Leishmaniasis (CL)* to splenomegaly and hepatomegaly in *visceral Leishmaniasis (VL)*, which is the most deadly form of *Leishmaniasis* (Henry W. Murray et al. 2005). According to the last report for *Leishmaniasis* emitted by *WHO*, it is estimated that annually there are about 1 million infections for *CL* and 90,000 for *VL* (“Leishmaniasis” n.d.). In fact, a small number of drugs are used for *Leishmaniasis* treatment (Roatt et al. 2020). Although these conventional therapies have some advantages, such as their efficacy, and have been used for several years, problems and disadvantages in their current usage have been documented, ranging from high costs, toxicity, and resistance by the parasite (Matos et al. 2020). This scenario leads us to the need to find more effective treatment mechanisms that present the least amount of disadvantages and problems to the patient due to their use.

Host-directed therapies (HDTs) are a group of strategies that interfere in the host mechanism that is necessary for pathogen survival, and stimulate the immune response in order to respond to pathogens mediated by the host, to eliminate them bypassing existing problems with conventional treatments, such less chance of developing resistance (Varikuti et al. 2018; Kaufmann et al. 2018). Recently *HDTs* have been proposed for the treatment of diverse bacterial, viral, and parasitic diseases (Zumla, Rao, Wallis, et al. 2016), such as *tuberculosis* (Zumla, Rao, Dodoo, et al. 2016), malaria (Smith et al. 2015), *HIV* infections (Ylisastigui et al. 2004; Matalon, Rasmussen, and Dinarello 2011), and most recently for the treatment of *SARS-CoV-2* (Zumla et al. 2020).

To this day, a variety of strategies has been developed in order to identify new host-directed therapies for *Leishmaniasis* treatment (Varikuti et al. 2018; Kumar et al. 2017). Many of these strategies are focused on the improvement of the immune response of the host (Kumar et al. 2017). In a previous work, Murray and colleagues demonstrated that a combination of *IL-12* and typical treatment with *pentavalent antimony (Sbv)* helped in the recovery of animals infected with *Leishmania donovani*, proving that a joint therapy between drugs that improved the host's immune response and conventional therapies can be useful for *Leishmaniasis* treatment (H. W. Murray et al. 2000). Another study discovered that *imatinib*, an anti-cancer drug, was useful for reducing the severity of lesions caused by *Leishmania amazonensis* (Wetzel, McMahon-Pratt, and Koleske 2012). Other studies were focused on promoting the production of *INF-γ* (Dubovsky et al. 2013), *NO* and IL-12 (Saha et al. 2011), *ROS* (Kyriazis et al. 2016) resulting in the improvement of immune response and promoting the healing and elimination of *Leishmania* parasites.

Networks are a form of knowledge representation used to structure relations between entities or objects. In biomedicine, the knowledge that is being represented and studied involves a high number of players with multiple connections. Biological networks are complex and very heterogeneous and contain a myriad of knowledge that has to be taken into consideration while analysing these networks. A perfect example of this complexity is cell networks, which consist of multiple sorts of biological molecules, connected in different means. To overcome this problem, the concept of multilayered graphs was introduced and employed in the field of biomedicine in the past years. This concept consists of layering the components of the network, nodes, or edges, by grouping them. The layering is based on the heterogeneity of the nodes of the network, or the relations between them. Colors or layers are assigned consequently to these elements, which results in two categories: node-colored graphs and edge-colored graphs, in which layers represent respectively the colors of the nodes and the edges. These aspects are employed in many fields in biomedicine, such as *protein-protein interaction (PPI)*, *gene regulatory networks (GRNs)*, and epidemiology (Hammoud and Kramer 2020a). Recently the use of network-based approaches to determine non-obvious biological interactions and their relationship to disease has been increasing (Conte et al. 2020). *Gene Regulatory Networks (GRNs)*, are a type of networks which represent regulatory events, between regulatory elements, such as *transcription factors (TFs)* or *non-coding RNAs (ncRNAs)* and genes (Walhout 2011). *GRNs* in combination with expression data can be used to infer condition-specific networks, which can be compared with control contexts or steady states, allowing the identification of possible disease markers or that can serve as possible targets for a pharmacological treatment (Sonawane et al. 2019).

Here we present a combination of *GRNs* based on transcriptomic data to model *Leishmania* infection dynamics in human macrophages and multilayered networks approach for the pathway mapping and drug-target identification approach in order to determine potential therapeutic targets for host-directed antileishmanial therapies. We identify five genes that have the potential as novel therapeutic targets. In addition to these potential targets, we describe the direct connection of 11 drugs, which could be repositioned to be used as host-directed antileishmanial therapy.

2. Materials and Methods

2.1. RNA-seq dataset of Leishmania infected macrophages

A set of *RNA-seq* derived from *Leishmania* major infected macrophage with bioproject accession *PRJNA290995* was downloaded. This data set is composed of 4 point times, 4 hours *post-infection (hpi),* 24 *hpi*, 48 *hpi*, and 72 *hpi*. In Figure 1, we resume the methods explained below.

2.2. RNA-seq data analysis

Raw data quality control was performed using *FastQC* ("Babraham Bioinformatics - FastQC A Quality Control Tool for High Throughput Sequence Data" n.d.) *v0.11.8*, low-quality reads were removed with *Trimmomatic 0.36* (Bolger, Lohse, and Usadel 2014) with a *Phred* cut-off value of $Q = 30$. Then, we mapped the remaining high quality reads with *Hisat2 version 2.1.0* (Kim, Langmead, and Salzberg 2015) to human genome version *GRCh38*. Expression values were calculated using *HTSeq-count 0.7.2* (Anders, Pyl, and Huber 2015). The resulting count reads were normalized and tested to identify the differentially expressed genes using *DESeq2* (Love, Huber, and Anders 2014). Genes with adjusted $p - value \leq 0.5$ and $absolut\ Log\ fold\ change \geq 0.5$ were considered as differentially expressed.

2.3. Gene regulatory network analyses

A human reference *gene regulatory network (GRN)* was obtained from *DoRothEA* (Garcia-Alonso et al. 2019) using only high-quality connections. This reference *GRN* was filtered using *RNA-seq* normalized data, as described before by Santander and cols. (Santander et al. 2018) with the following modifications: interaction between *transcription factor (TF)* and their target were only saved if normalized reads value for *TF* were at least 10. After we obtained filtered networks for all conditions, we applied a pairwise comparison based on infected against non-infected for all four serial times, using *LoTo* (Martin et al. 2017). Once these comparisons were obtained, we evaluated the *F1* values of all *TFs* and *non-TFs* genes. This value ranges from 0 to 1, when 1 represents a higher similarity of X node in both networks. After the evaluation of this metric, we saved only nodes and edges that appeared in the infected but not in non-infected networks. A preliminary list of interest genes was filtered using an absolute *logFC* of 0.5 and genes were selected only if they participated in processes related to immune response, response to stress, or host-pathogen interaction.

2.4. Gene-protein and protein-pathway mapping and drug-target interaction

Multipath is an *R* package to generate integrated reproducible pathway knowledge (Hammoud and Kramer 2020b). Using *Multipath*, *BioPAX*-encoded pathways can be modelled into multilayered graphs, where the biological pathways' components are embedded into different layers based on their biological type. The built graphs are reproducible, i.e. all modifications applied to the graphs are stored. *Multipath* is also used to integrate influencing pathway knowledge from external databases like drugs from *DrugBank* (Wishart et al. 2018). We used this package to query pathway knowledge databases and fetch relevant information needed in our computational analysis. To map the gene set of interest to their gene products from *UniProt*, *Multipath* uses *UniProt.ws* to fetch the *UniProt* IDs of the corresponding proteins, mapped to our list of 113 *HGNC* symbols. Then we got a list of biological pathways from *Reactome* (Jassal et al. 2020) in which these proteins participate. These *Reactome* IDs were downloaded to generate *mully* (Hammoud and Kramer 2018) multilayered graphs. Next, we filtered all proteins that were not included in our gene list. Finally, we extracted the drug targets from *UniProt* and *DrugBank*, and added a drug layer to each filtered pathway graph, and only preserved gene products to which direct drug connections were identified.

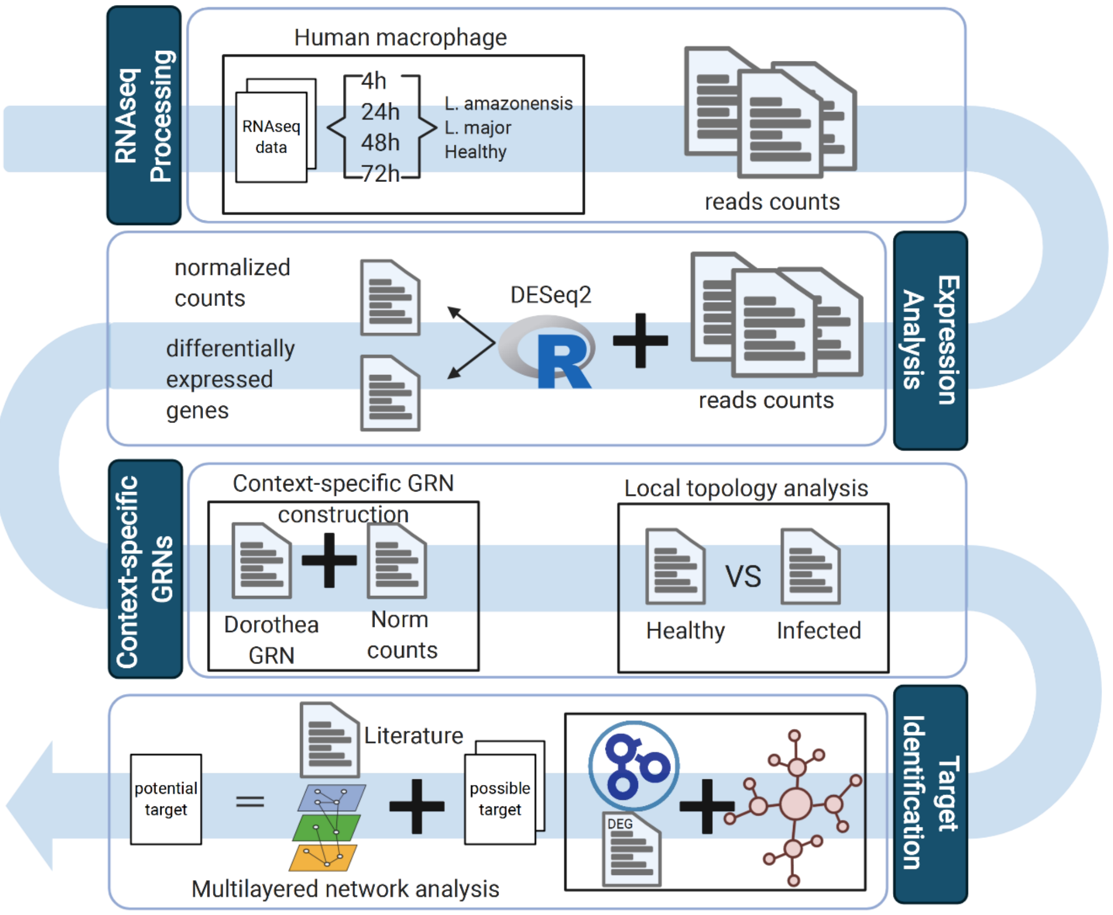


Figure 1. Pipeline to identify potential therapeutic target for *Leishmaniasis* host-directed treatment in human macrophage from *RNA-seq* data.

3. Results

3.1. Global expression patterns in Leishmania major infected macrophages

A global gene expression analysis was performed using a publicly available set of *RNA-seq* data comprising four time points (4, 24, 48, 72 hpi) of *L. major* infected human macrophages previously reported by Fernandes and cols. (Fernandes et al. 2016).

Transcriptomic analysis reveals a high number of *differentially expressed genes (DEGs)* in paired-sample analysis (health against infected macrophage) at the first hours after infection (Figure 3). At 4 *hpi* we observed a higher number of *DEGs*, at this time post-infection 4709 genes were identified as *DEGs* based on $FDR \leq 0.05$ and $-0.5 > FC > 0.5$. For this set of *DEGs* we obtained that 2521 were upregulated and 2188 downregulated in the infected macrophage compared to the health macrophage. On the other hand, we showed that *DEGs* decreased in order to hours post-infection increase, presenting a lower number of *DEGs* at 72 *hpi* with a total of 953 *DEGs* of which 413 was downregulated and 540 upregulated (Figure 3).

Based on the set of *DEGs*, we applied a *GO* enrichment analysis to identify biological processes associated with host-pathogen interaction, response to stress, and immune response. Enriched *GO* categories obtained for up- and downregulated *DEGs* from health against infected macrophage comparison are listed in Supplementary data set 1. Interestingly, three *GO* categories were upregulated at the 4 *hpi* related to response to *cytokine (GO:0034097)*, cellular response to *cytokine stimulus (GO:0071345)*, and *cytokine-mediated signaling pathway (GO:0019221)*. Moreover, regulation of response to *stress (GO:0019221)*, Regulation of response to *stress*, Negative regulation of response to *stimulus*, Positive regulation of response to *stimulus*, also was upregulated at 4 *hpi*. On the other hand, we didn't find any enriched *GO* term downregulated related to immune response, host-pathogen interaction, or response to *stress* (Supplementary data set 1). Similar to 4 *hpi*, we identified that *cytokine* related terms were upregulated in 24 *hpi*, likewise, the stimulus terms also were identified as upregulated (Supplementary data set 1). At 48 *hpi* we only obtained response to *cytokine (GO:0034097)*, cellular response to *cytokine stimulus (GO:0071345)* as upregulated, and any other term related to *stress* or response to *stimulus* were identified as enriched. Finally, our enrichment analysis showed that Response to *cytokine (GO:0034097)* but no other *cytokine* related term was enriched (Supplementary data set 1). Additionally, terms related to *ncRNA metabolism*, such as *ncRNA processing (GO:0034470)*, *ncRNA metabolic process (GO:0034660)*, were consistently enriched in 24, 48, and 732 *hpi* (Supplementary data set 1). This corresponds with the higher number of *lncRNA* and *antisense RNAs* differentially expressed (Supplementary Figure 1).

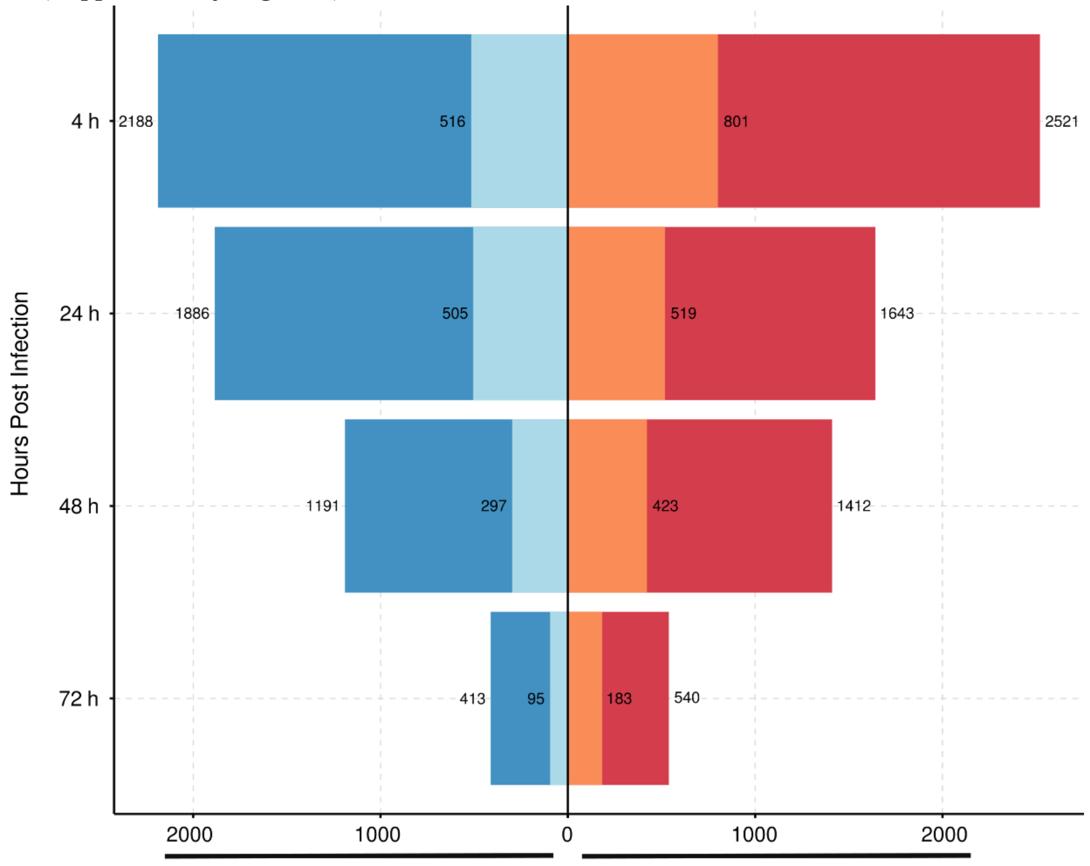


Figure 2. Global transcriptomic profiles of *Leishmania* infected human macrophages and genes related to immune response and host-pathogen interaction. Distribution of *DEGs* between different specific time post-infection. The box width indicates the number of *DEGs* downregulated (blue) and upregulated (red) at adjusted *p-value* 0.05 and -0.5 > *FC* > 0.5. The color shading indicates *DEGs* that are involved in host-pathogen interaction and immune response according to *GO* functional analysis. The numbers at the end of each bar correspond to total *DEGs* obtained after paired-samples analysis.

Due to the low number of enriched *GO* terms in roles of immune response or host-pathogen interaction, we performed a functional analysis in order to find those genes that had some annotation related to immune response processes and host-pathogen interaction. As we showed in Supplementary Table 1 and Figure 2, we identified between 183 to 801 upregulated genes and related to immune response, host-pathogen functional categories, and between 95 to 516 downregulated genes annotated with these functional categories. The identity of these genes was deposited in the supplementary data set 2.

Interestingly as we found for *DE* the number of related genes with immune response and host-pathogen interaction decreases with the passing of the hours after infection.

3.2. Context-specific gene regulatory networks of Leishmania infected macrophage reveal potential new therapeutic targets

The *TF-gene reference human gene regulatory network (GRN)* obtained after filter high-quality connections are described in table 1. After that, we used the normalized reads obtained from our previous expression analysis to filter this gold standard *GRN*. We obtained eight time-specific *GRNs* (four of infected and four of non-infected macrophages) that illustrate the dynamics of gene regulations at the first 72h after the infection process of *Leishmania* parasites begins. Each context-specific network presents a different number of nodes and connections described in table 1. The larger network for infected macrophage was obtained at 4 *hpi* that includes 19750 total nodes and 343072 connections with 990 *TFs* as regulatory nodes. Besides, the smaller network of infected macrophage corresponding to 72 *hpi* made up with 19718 nodes, of which 974 nodes were identified as *TFs* and recovered a total of 339390 connections.

	Total genes	**Edges**	**TFs**
Reference GRN (DOROTHEA)	20244	486751	13950990
4 h post infection	19750	343072	990
24 h post-infection	19750	343201	987
48 h post-infection	19725	341608	987
72 h post-infection	19718	339390	974
4 h non-infected	19760	344812	999
24 h non-infected	19748	342770	993
48 h non-infected	19595	342732	992
72 h non-infected	19610	345596	998

Table 1. Description of the reference network and context-specific *GRNs* of *Leishmania major* infected macrophages and non-infected macrophages.

We compare each network with their corresponding for the same time period. The *F1* values that we obtained from these comparisons ranged between 0.96 to 0.98, indicating that in general there are few alterations in the genetic regulation networks that are affected in the first hours after the infection of human macrophages.

We also analyze all *TF* that have a significant change of regulation according to *F1* values, for this, we use 0.95 *F1* value as cutoff (Table 2). These results indicate that many *TFs* related to immune response shows alterations in their regulations, besides not all these *TFs* were differentially expressed according to our expression analysis. Therefore, we selected all nodes and edges that were present in the infected macrophage networks but not in the health macrophage networks (Figure 2A, Supplementary Figure 2A, Supplementary Figure 3A, Supplementary Figure 4B). Figure 2B shows the subnetwork after filtering all the edges only present in the infected network at 4 *hpi* where we identified 160 connections and 63 nodes of which 11 were *TF*. On the other hand, we found 244 nodes for 24 *hpi* network comparison (Supplementary Figure 2B), 155 for 48 hpi (Supplementary Figure 3B), and 52 for 72 *hpi* (Supplementary Figure 4B). In table 3 we resume all these findings.

All genes recovered from these comparisons were evaluated in order to select all those genes that were differentially expressed and related to the functions that we have previously described. A total of 373 genes were finally selected of which 113 passed all filters that we applied. Interestingly we identified several *TF* that act in the activation of immune response such as *JUN* or the negative regulation of immune response such as *MYC*. On the other hand, we determine that some effectors of immune response such as *IL16*, that act as pro-inflammatory were present in our list of possible therapeutic targets. In Supplementary data set 3, we summarise all the identities, expression values, and functions of all 113 genes.

Comparison	**TF**	**F1**	**Function**
4 h	MEF2B	0	-
	PROX1**	0	Regulation of developmental process
	KLF1**	0.339	Immune system process
	E2F2**	0.611	Regulation of developmental process
	FLI1**	0.931	Immune system process
	STAT4*	0.934	-
	TCF3	0.935	Immune system process/Leukocyte activation
24 h	MEF2B	0	-
	NFE2	0	Immune system process
	PROX1**	0	Regulation of developmental process
	POU5F1	0.019	Response to stress
	TCF7**	0.078	Immune system process/Leukocyte activation
	ATF6	0.536	Response to stress
	CEBPD**	0.850	Immune system process
	NCOA2**	0,853	Response to endogenous stimulus
	TCF3	0.899	Immune system process/Leukocyte activation
	ZEB1	0.907	Immune system process/Leukocyte activation
	RUNX3	0.918	Immune system process/Leukocyte activation
	FLI1**	0.930	Immune system process
	FOXO3	0.937	Immune system process
	EPAS1	0.949	Immune system process
48 h	MEF2B	0	-
	POU5F1	0.019	Response to stress
	TBX21	0.245	Immune system process/Leukocyte activation
	SOX6	0.412	Immune system process
	TP73**	0.478	Immune system process
	CEBPD**	0.854	Immune system process
	FLI1**	0.929	Immune system process
	ZEB2	0.930	Response to stress
	STAT4	0.935	-
	TCF3	0.935	Immune system process/Leukocyte activation
	TCF4	0.942	Regulation of response to stimulus
72 h	MEF2B	0	-
	POU5F1	0.019	Response to stress
	TCF7**	0.021	Immune system process/Leukocyte activation
	TBX21	0.245	Immune system process/Leukocyte activation
	ELF3	0.335	Response to stress
	SOX6	0.362	Immune system process
	CEBPD	0.849	Immune system process
	RUNX3	0.912	Immune system process/Leukocyte activation
	IKZF1	0.917	Immune system process/Leukocyte activation
	ZEB2	0.932	Response to stress
	STAT4	0.934	-
	TCF3	0.936	Immune system process/Leukocyte activation

Table 2. TFs with higher changes in their regulations in Leishmania major infected macrophages.

We used these 113 genes to obtain the biological pathways, which the product of this set of genes participates. We obtained a list of 909 *UniProt* IDs of protein entries in *UniProt* mapped to 313 *Reactome* IDs of *Reactome* pathway entries, which we downloaded and used to generate 313 *mully* multilayered graphs. After mapping the proteins on the protein layers to our list of proteins, the *mully* graphs were filtered by deleting all non-protein layers, while adding the transitive edges to preserve the connections.

Comparison	**Edge**	**Total nodes**	**Nodes as TFs**
4 hpi	160	63	11
24 hpi	246	244	21
48 hpi	154	155	20
72 hpi	50	52	9

Table 3. Nodes and edges only present in Leishmania major infected macrophage specific context networks.

We also deleted the proteins that are not included in our list. We extracted the drug targets from *UniProt* and *DrugBank* and added a drug layer to each filtered pathway graph. We combined all drug-protein connections from the different graphs to obtain the final list of possible drug targets (Figure 4). A final set of 21 gene-pathway-drug interactions were obtained (Supplementary data set 4). We identified 124 different biological pathways and 331 different drugs that have direct connections with these 21 genes. In order to reduce the number of gene-drug interactions, we filtered these interactions, using as selection criteria those drugs that were approved, as well as any of the following combinations: approved-investigational and approved-vet-approved. A total of 195 approved drugs were finally selected, these drugs directly interact with 13 different genes.

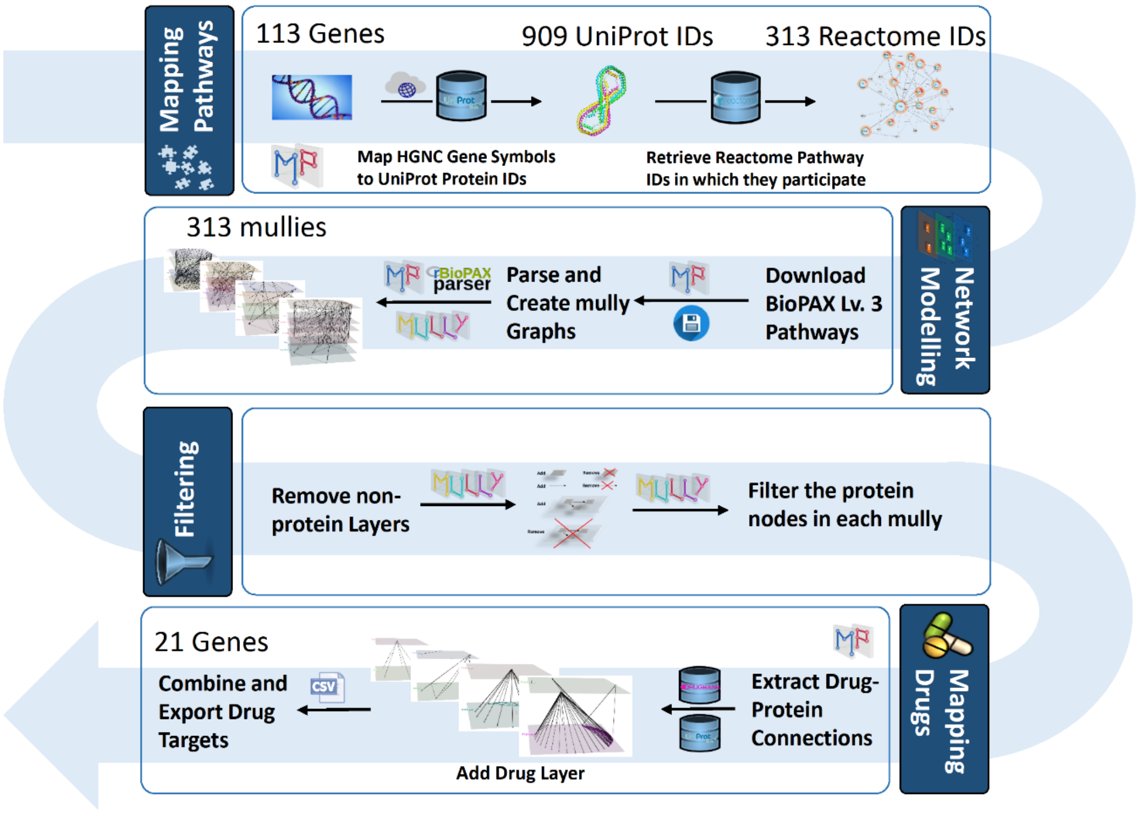

Mapping Pathways
113 Genes
909 UniProt IDs
313 Reactome IDs
Map HGNC Gene Symbols to UniProt Protein IDs
Retrieve Reactome Pathway IDs in which they participate
313 mullies
BioPAX parser
Parse and Create mully Graphs
Download BioPAX Lv. 3 Pathways
Network Modelling
Filtering
Remove non-protein Layers
Filter the protein nodes in each mully
21 Genes
Combine and Export Drug Targets
Add Drug Layer
Extract Drug-Protein Connections
Mapping Drugs

Figure 4. Workflow Diagram to identify the Drug Targets using the Multipath package.

After this selection, we evaluate these 13 genes in order to identify their relationship with *Leishmania* infection. Interestingly we found 8 genes that were literature confirmed as genes involved in *Leishmania* infection (Veras, Ramos, and de Menezes 2018; Osorio et al. 2014; Barrera et al. 2017; Islamuddin et al. 2016; Sellau, Groneberg, and Lotter 2019). In Supplementary table 2, we include all 8 final genes of potential candidates to *host-directed* therapeutic targets. We found that these 8 genes have a direct connection to 145 different drugs. This information was crossed with our expression data, toxicity and previous used to determine the best set of different drugs that could be repositioned for use as *host-directed* therapy for the treatment of *leishmaniasis*.

Target	**Drug**	**DrugBank ID**	**Pharmacological Action**	**Action**	**Uses**	**Antileishmanial evidence**
AR	Esculin	DB13155	Unknown	Agonist	Vasoprotective agent	No
	Clascoterone	DB12499	Yes	Antagonist	Acne	No
	Acetophenazine	DB01063	Unknown	Unknown	Antipsychotic	No
JUN	Adapalene	DB00210	Yes	Antagonist	Acne	No
PDGFRA	Ripretinib	DB14840	Yes	Inhibitor	Antineoplastic	No
PTGS2	Tolfenamic acid	DB09216	Yes	Antagonist	Non-steroidal anti-inflammatory drug (nsaid)	No
	Flufenamic acid	DB02266	Unknown	Unknown	Non-steroidal anti-inflammatory drug (nsaid)	No
	Antrafenine	DB01419	Unknown	Inhibitor	Analgesic	No
VEGFA	Dalteparin	DB06779	Yes	Inhibitor	Trombosis	No
	Minocycline	DB01017	Unknown	Inhibitor	Antibiotic	No
	Pidolic acid	DB03088	Unknown	Unknown	Moisturizer for dry skin	No

Table 4. Potential therapeutic targets for *host-directed Leishmaniasis* treatment and their best drug connection.

Finally, we obtained five genes with a high potential to be a new therapeutic target, genes such as *Androgen receptor (AR)* or *Prostaglandin-endoperoxide synthase 2 (PTGS2)* or *Vascular endothelial growth factor A (VEGFA)* have at least three drugs each that could be used as *host-directed* therapies for *Leishmaniasis* treatment (Table 4). Interestingly some of these drugs such as *Clascoterone* and *Adapalene* have previously been used as drugs to treat skin conditions such as acne.

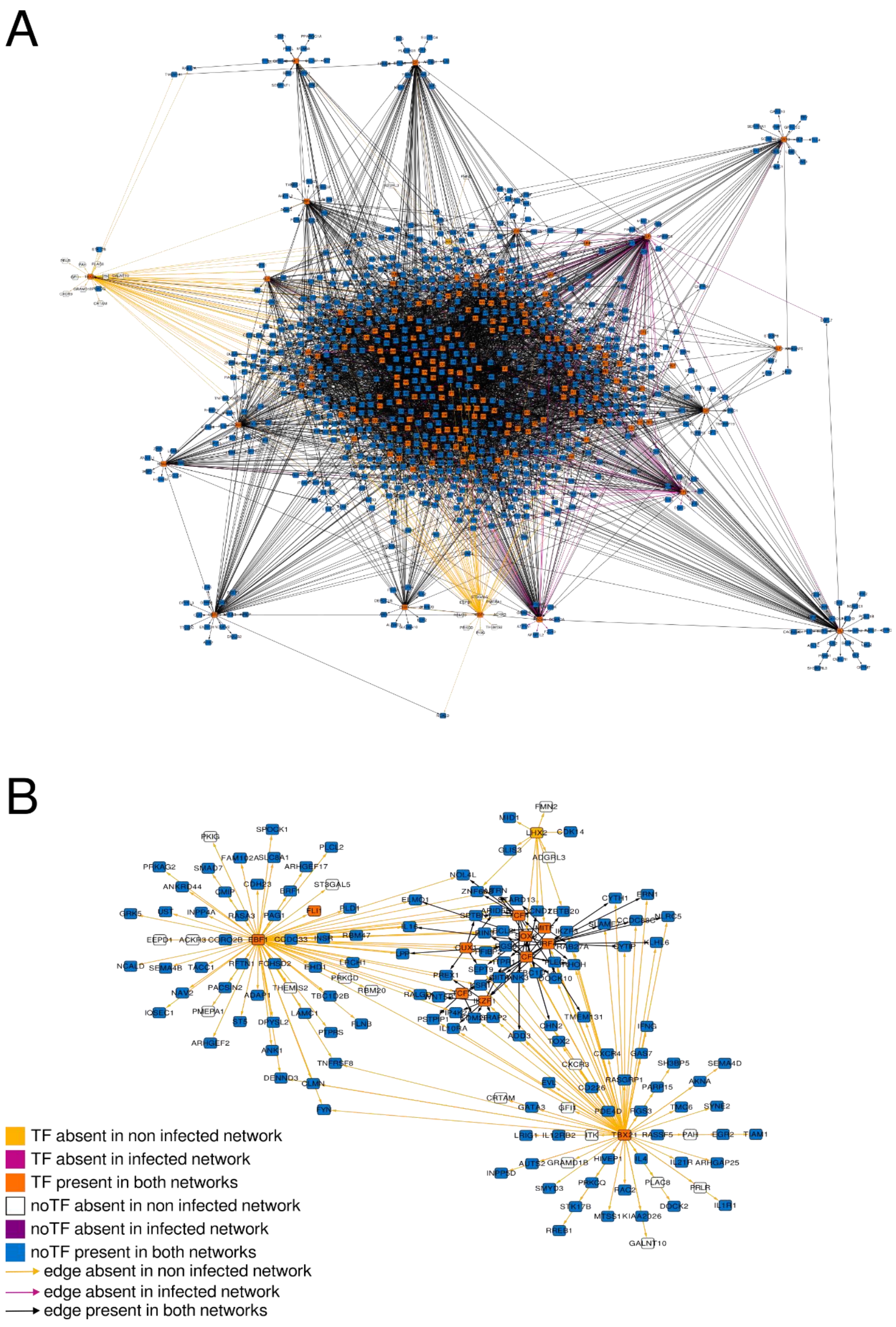

A
B
TF absent in non infected network
TF absent in infected network
TF present in both networks
noTF absent in non infected network
noTF absent in infected network
noTF present in both networks
edge absent in non infected network
edge absent in infected network
edge present in both networks

Figure 3. Network comparison of non infected against infected macrophage at 4 hours post infection. A. The network shown is formed by 942 nodes (167 *TFs*) and 3847 edges colored according to their existence in the non-infected network, infected network or in both networks. **B.** Subnetwork represents all edges presented only in the 4 *hpi* network. Colors of edges and nodes are the same as the upper network.

On the other hand, anti-inflammatories like *Tolfenamic acid* and Flufenamic acid are also potential drugs for the treatment of *Leishmaniasis*. Our results indicated that several new therapeutic targets could be identified from the changes in gene regulation that occur during the infective process of *Leishmania major* to human macrophages and the integration of data related to pathways and gene-drug direct connections.

4. Discussion

Conventional therapies for *Leishmaniasis* treatment typically are composed of only a few numbers of drugs (Sundar and Singh 2018; Chakravarty and Sundar 2019). Moreover, these therapies present a great variety of problems mainly in the efficiency, toxicity, and the ability of parasites to generate resistance to them (Matos et al. 2020). Recently, different approaches for the treatment of several parasitic diseases were focused on the *host-directed* therapies (Varikuti et al. 2018) and the repositioning of drugs used previously for another disease (Andrews, Fisher, and Skinner-Adams 2014). Here, we demonstrated that the use of network-based approaches is a powerful tool to identify new therapeutic targets for *host-directed Leishmaniasis* treatment.

In this study, we found 5 genes with the potential to be new therapeutic targets. We identified that these genes have direct connections with eleven *FDA*-approved drugs (Table 4). Many of these drugs have previously been used to treat skin diseases such as acne, anti-inflammatory drugs, or *antineoplastics*. Drugs such as *Clascoterone* act as antagonists for *androgen receptor (AR)*. Previous reports have shown that *AR* has a very important role in the immune response derived from parasitic infections (Sellau, Groneberg, and Lotter 2019). Interestingly, in two different works reported by Sanchez-Garcia and colleagues and Qiao and colleague discovered that hormones such *DHT* and *testosterone* that interact with the *androgen receptor*, can alter the development and the survival of the parasite, or inhibit the *apoptosis* of *Leishmania*-infected macrophages (Sánchez-García et al. 2018; Qiao et al. 1999). This suggests that *androgen receptor* is a very promising target and their blocking by some an *antagonist* drug plus the implementation of conventional treatments could be of help as a treatment for *Leishmania* infections.

In an effort to know more about our selected potential new therapeutic targets, we found that *prostaglandin endoperoxide synthase 2 (PTGS2)* previously was reported as a biomarker in response to infections for *L. major* (Veras, Ramos, and de Menezes 2018). Our results show a consistent misregulation of this gene consistently with these previous studies. Also, we identified a great number of direct connections with different kinds of drugs, such as *Tolfenamic acid a NSAID* that can be used effectively for the treatment of *schistosomiasis* (Lago et al. 2019). This indicates that possibly the repositioning of this drug could be useful as an antileishmanial. However, an *in vivo* testing needs to be done in order to confirm that this repositioning would be useful.

Antitumoral drugs previously were used to treat *Leishmaniasis*; *miltefosine* was developed as an anticancer drug, but today is a choice for *visceral Leishmaniasis* treatment (Andrade-Neto et al. 2018). *Ripretinib* was described as a promising drug for the treatment of gastrointestinal stromal tumors and more recently, their drug family has been proposed to be repositioned for *COVID-19* treatment (Catalano et al. 2021). The repositioning of *Ripretinib* will be useful for *Leishmaniasis* treatment. This is because *Ripretinib* acts as a *kinase inhibitor* and presents an inhibition activity over *platelet derived growth factor receptor (PDGFRA) a te tyrosine kinase receptor* that has been targeted for antileishmanial therapies, presenting a significant reduction in parasitic survival (Sanderson, Yardley, and Croft 2014).

Vascular endothelial growth factor-A (VEGFA) is induced during *Leishmania major* infection (Weinkopff et al. 2019, 2016). Their expression promotes *lymphangiogenesis* that was implicated in the inflammatory response and lesion healing (Weinkopff et al. 2019). *VEGFA* was used as a target in antiangiogenesis therapies in order to reduce the vascularization of tumors (Ferrara 2005). *Minocycline* is an antibiotic from the *tetracycline* family that interacts with the *VEGFA* and inhibits angiogenesis (Jung et al. 2014). This antibiotic was proposed to be used for parasitic infection in the late 80’s mainly for the treatment of *Giardia lamblia* where a potent activity was observed to reduce the survival of the parasite *in vitro* (Edlind 1989). On the other hand in a more recent study, a patient diagnosed with *Leishmaniasis* was treated with *Minocycline*, however, the treatment was stopped shortly after, and no conclusive results were obtained (Cargnelutti et al. 2016).

JUN gene encodes a *c-Jun* protein that is a central part of *AP-1* transcription factor. This transcription factor is crucial for the inactivation of macrophages during *Leishmania* infection (Contreras et al. 2010). *Adapalene* is a *retinoid* mainly used for acne treatment and interacts with *c-JUN* via *AP-1* transcription factor (Liao et al. 2020). Interestingly, *Adapalene* was used as an antitumoral drug in an *in vitro* assay promoting the *apoptosis* of colorectal cells (Ocker et al. 2003). However, its antiparasitic activity has not been proven yet, thus requiring an experimental validation to trace its effectiveness as an antileishmanial drug.

Our approach for the search of new therapeutic targets for *host-directed* antileishmanial strategy provides a significant number of novel potential targets. This strategy helps to bypass the problems arising from conventional antiparasitic therapies. Furthermore, we demonstrate in this study that many drugs could be repositioned for *Leishmaniasis* treatment. However, all drugs selected in our work remain to be experimentally tested to know their potential as *host-directed* antileishmanial therapies. Finally, our approach shows great potential for the identification of therapeutic targets from genomic information and data analysis derived from biological information, which can be applied not only to parasitic diseases but also to a great variety of infectious diseases.

5. References

Anders, Simon, Paul Theodor Pyl, and Wolfgang Huber. 2015. “HTSeq--a Python Framework to Work with High-Throughput Sequencing Data.” Bioinformatics 31 (2): 166–69.

Andrade-Neto, Valter Viana, Edezio Ferreira Cunha-Junior, Viviane Dos Santos Faioes, Thaís Martins Pereira, Raphaela Lopes Silva, Leonor Laura Leon, and Eduardo Caio Torres-Santos. 2018. “Leishmaniasis Treatment: Update of Possibilities for Drug Repurposing.” Frontiers in Bioscience 23 (January): 967–96.

Andrews, Katherine T., Gillian Fisher, and Tina S. Skinner-Adams. 2014. “Drug Repurposing and Human Parasitic Protozoan Diseases.” International Journal for Parasitology, Drugs and Drug Resistance 4 (2): 95–111.

“Babraham Bioinformatics - FastQC A Quality Control Tool for High Throughput Sequence Data.” n.d. Accessed February 26, 2020. https://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

Barrera, Maria Claudia, Laura Jimena Rojas, Austin Weiss, Olga Fernandez, Diane McMahon-Pratt, Nancy G. Saravia, and Maria Adelaida Gomez. 2017. “Profiling Gene Expression of Antimony Response Genes in Leishmania (Viannia) Panamensis and Infected Macrophages and Its Relationship with Drug Susceptibility.” Acta Tropica 176 (December): 355–63.

Bolger, Anthony M., Marc Lohse, and Bjoern Usadel. 2014. “Trimmomatic: A Flexible Trimmer for Illumina Sequence Data.” Bioinformatics 30 (15): 2114–20.

Burza, Sakib, Simon L. Croft, and Marleen Boelaert. 2018. “Leishmaniasis.” The Lancet. https://doi.org/10.1016/s0140-6736(18)31204-2.

Cargnelutti, Diego Esteban, Carlos Guillermo Borremans, Rosa Lydia Tonelli, Liliana Carmen Carrizo, and María Cristina Salomón. 2016. “Diagnosis of Leishmania Infection in a Nonendemic Area of South America.” Journal of Microbiology, Immunology, and Infection = Wei Mian Yu Gan Ran Za Zhi 49 (5): 809–12.

Catalano, Alessia, Domenico Iacopetta, Michele Pellegrino, Stefano Aquaro, Carlo Franchini, and Maria Stefania Sinicropi. 2021. “Diarylureas: Repositioning from Antitumor to Antimicrobials or Multi-Target

Agents against New Pandemics.” Antibiotics (Basel, Switzerland) 10 (1). https://doi.org/10.3390/antibiotics10010092.

Chakravarty, Jaya, and Shyam Sundar. 2019. “Current and Emerging Medications for the Treatment of Leishmaniasis.” Expert Opinion on Pharmacotherapy 20 (10): 1251–65.

Conte, Federica, Giulia Fiscon, Valerio Licursi, Daniele Bizzarri, Tommaso D’Antò, Lorenzo Farina, and Paola Paci. 2020. “A Paradigm Shift in Medicine: A Comprehensive Review of Network-Based Approaches.” Biochimica et Biophysica Acta, Gene Regulatory Mechanisms 1863 (6): 194416.

Contreras, Irazú, María Adelaida Gómez, Oliver Nguyen, Marina T. Shio, Robert W. McMaster, and Martin Olivier. 2010. “Leishmania-Induced Inactivation of the Macrophage Transcription Factor AP-1 Is Mediated by the Parasite Metalloprotease GP63.” PLoS Pathogens 6 (10): e1001148.

Dubovsky, Jason A., Kyle A. Beckwith, Gayathri Natarajan, Jennifer A. Woyach, Samantha Jaglowski, Yiming Zhong, Joshua D. Hessler, et al. 2013. “Ibrutinib Is an Irreversible Molecular Inhibitor of ITK Driving a Th1-Selective Pressure in T Lymphocytes.” Blood 122 (15): 2539–49.

Edlind, T. D. 1989. “Tetracyclines as Antiparasitic Agents: Lipophilic Derivatives Are Highly Active against Giardia Lamblia in Vitro.” Antimicrobial Agents and Chemotherapy 33 (12): 2144–45.

Fernandes, Maria Cecilia, Laura A. L. Dillon, Ashton Trey Belew, Hector Corrada Bravo, David M. Mosser, and Najib M. El-Sayed. 2016. “Dual Transcriptome Profiling of Leishmania-Infected Human Macrophages Reveals Distinct Reprogramming Signatures.” mBio 7 (3). https://doi.org/10.1128/mBio.00027-16

Ferrara, Napoleone. 2005. “VEGF as a Therapeutic Target in Cancer.” Oncology 69 Suppl 3 (November): 11–16.

Garcia-Alonso, Luz, Christian H. Holland, Mahmoud M. Ibrahim, Denes Turei, and Julio Saez-Rodriguez. 2019. “Benchmark and Integration of Resources for the Estimation of Human Transcription Factor Activities.” Genome Research 29 (8): 1363–75.

Hammoud, Zaynab, and Frank Kramer. 2018. “: An R Package to Create, Modify and Visualize Multilayered Graphs.” Genes 9 (11). https://doi.org/10.3390/genes9110519

———. 2020a. “Multilayer Networks: Aspects, Implementations, and Application in Biomedicine.” Big Data Analytics 5 (1): 2.

———. 2020b. “Multipath: An R Package to Generate Integrated Reproducible Pathway Models.” Biology 9 (12). https://doi.org/10.3390/biology9120483

Islamuddin, Mohammad, Garima Chouhan, Muzamil Yaqub Want, Hani A. Ozbak, Hassan A. Hemeg, and Farhat Afrin. 2016. “Immunotherapeutic Potential of Eugenol Emulsion in Experimental Visceral Leishmaniasis.” PLoS Neglected Tropical Diseases 10 (10): e0005011.

Jassal, Bijay, Lisa Matthews, Guilherme Viteri, Chuqiao Gong, Pascual Lorente, Antonio Fabregat, Konstantinos Sidiropoulos, et al. 2020. “The Reactome Pathway Knowledgebase.” Nucleic Acids Research 48 (D1): D498–503.

Jung, Hui-Jung, Incheol Seo, Bijay Kumar Jha, Seong-Il Suh, Min-Ho Suh, and Won-Ki Baek. 2014. “Minocycline Inhibits Angiogenesis in Vitro through the Translational Suppression of HIF-1α.” Archives of Biochemistry and Biophysics 545 (March): 74–82.

Kaufmann, Stefan H. E., Anca Dorhoi, Richard S. Hotchkiss, and Ralf Bartenschlager. 2018. “Host-Directed Therapies for Bacterial and Viral Infections.” Nature Reviews. Drug Discovery 17 (1): 35–56.

Kim, Daehwan, Ben Langmead, and Steven L. Salzberg. 2015. “HISAT: A Fast Spliced Aligner with Low Memory Requirements.” Nature Methods 12 (4): 357–60.

Kumar, Rajiv, Shashi Bhushan Chauhan, Susanna S. Ng, Shyam Sundar, and Christian R. Engwerda. 2017. "Immune Checkpoint Targets for Host-Directed Therapy to Prevent and Treat Leishmaniasis." Frontiers in Immunology 8 (November): 1492.

Kyriazis, Ioannis D., Olga S. Koutsoni, Nektarios Aligiannis, Kalliopi Karampetsou, Alexios-Leandros Skaltsounis, and Eleni Dotsika. 2016. "The Leishmanicidal Activity of Oleuropein Is Selectively Regulated through Inflammation- and Oxidative Stress-Related Genes." Parasites & Vectors 9 (1): 441.

Lago, Eloi M., Marcos P. Silva, Talita G. Queiroz, Susana F. Mazloum, Vinícius C. Rodrigues, Paulo U. Carnaúba, Pedro L. Pinto, et al. 2019. "Phenotypic Screening of Nonsteroidal Anti-Inflammatory Drugs Identified Mefenamic Acid as a Drug for the Treatment of Schistosomiasis." EBioMedicine 43 (May): 370–79.

"Leishmaniasis." n.d. Accessed December 14, 2020. https://www.who.int/en/news-room/fact-sheets/detail/leishmaniasis.

Liao, Ai-Ho, You-Lin Cai, Ho-Chaio Chuang, Cheng-Ying Lee, Yu-Chun Lin, and Chien-Ping Chiang. 2020. "Application of Ultrasound-Mediated Adapalene-Coated Lysozyme-Shelled Microbubbles in UVA-Induced Skin Photoaging." PloS One 15 (5): e0232617.

Love, Michael I., Wolfgang Huber, and Simon Anders. 2014. "Moderated Estimation of Fold Change and Dispersion for RNA-Seq Data with DESeq2." Genome Biology. https://doi.org/10.1186/s13059-014-0550-8.

Martin, Alberto J., Sebastián Contreras-Riquelme, Calixto Dominguez, and Tomas Perez-Acle. 2017. ": A Graphlet Based Method for the Comparison of Local Topology between Gene Regulatory Networks." PeerJ 5 (February): e3052.

Matalon, Shay, Thomas A. Rasmussen, and Charles A. Dinarello. 2011. "Histone Deacetylase Inhibitors for Purging HIV-1 from the Latent Reservoir." Molecular Medicine 17 (5-6): 466–72.

Matos, A. P. S., A. L. Viçosa, M. I. Ré, E. Ricci-Júnior, and C. Holandino. 2020. "A Review of Current Treatments Strategies Based on Paromomycin for Leishmaniasis." Journal of Drug Delivery Science and Technology. https://doi.org/10.1016/j.jddst.2020.101664.

Murray, Henry W., Jonathan D. Berman, Clive R. Davies, and Nancy G. Saravia. 2005. "Advances in Leishmaniasis." The Lancet 366 (9496): 1561–77.

Murray, H. W., C. Montelibano, R. Peterson, and J. P. Sypek. 2000. "Interleukin-12 Regulates the Response to Chemotherapy in Experimental Visceral Leishmaniasis." The Journal of Infectious Diseases 182 (5): 1497–1502.

Ocker, Matthias, Christoph Herold, Marion Ganslmayer, Eckhart G. Hahn, and Detlef Schuppan. 2003. "The Synthetic Retinoid Adapalene Inhibits Proliferation and Induces Apoptosis in Colorectal Cancer Cells in Vitro." International Journal of Cancer. Journal International Du Cancer 107 (3): 453–59.

Osorio, E. Yaneth, Bruno L. Travi, Alda M. da Cruz, Omar A. Saldarriaga, Audrie A. Medina, and Peter C. Melby. 2014. "Growth Factor and Th2 Cytokine Signaling Pathways Converge at STAT6 to Promote Arginase Expression in Progressive Experimental Visceral Leishmaniasis." PLoS Pathogens 10 (6): e1004165.

Qiao, Z., Z. Guo, G. Yin, L. Yin, J. Zhao, and F. Wunderlich. 1999. "Testosterone Inhibits Apoptosis of Leishmania Donovani-Infected Macrophages." Zhongguo Ji Sheng Chong Xue Yu Ji Sheng Chong Bing Za Zhi = Chinese Journal of Parasitology & Parasitic Diseases 17 (1): 21–24.

Roatt, Bruno Mendes, Jamille Mirelle de Oliveira Cardoso, Rory Cristiane Fortes De Brito, Wendel Coura-Vital, Rodrigo Dian de Oliveira Aguiar-Soares, and Alexandre Barbosa Reis. 2020. "Recent

Advances and New Strategies on Leishmaniasis Treatment." Applied Microbiology and Biotechnology 104 (21): 8965–77.

Saha, Piu, Surajit Bhattacharjee, Avijit Sarkar, Alak Manna, Subrata Majumder, and Mitali Chatterjee. 2011. "Berberine Chloride Mediates Its Anti-Leishmanial Activity via Differential Regulation of the Mitogen Activated Protein Kinase Pathway in Macrophages." PloS One 6 (4): e18467.

Sánchez-García, L., A. Wilkins-Rodriguez, N. Salaiza-Suazo, J. Morales-Montor, and I. Becker. 2018. "Dihydrotestosterone Enhances Growth and Infectivity of Leishmania Mexicana." Parasite Immunology 40 (3). https://doi.org/10.1111/pim.12512.

Sanderson, Lisa, Vanessa Yardley, and Simon L. Croft. 2014. "Activity of Anti-Cancer Protein Kinase Inhibitors against Leishmania Spp." The Journal of Antimicrobial Chemotherapy 69 (7): 1888–91.

Santander, Nicolás, Carlos Lizama, Leandro Murgas, Sebastián Contreras, Alberto J. M. Martin, Paz Molina, Alonso Quiroz, et al. 2018. "Transcriptional Profiling of Embryos Lacking the Lipoprotein Receptor SR-B1 Reveals a Regulatory Circuit Governing a Neurodevelopmental or Metabolic Decision during Neural Tube Closure." BMC Genomics 19 (1): 731.

Sellau, Julie, Marie Groneberg, and Hannelore Lotter. 2019. "Androgen-Dependent Immune Modulation in Parasitic Infection." Seminars in Immunopathology 41 (2): 213–24.

Smith, Clare M., Ante Jerkovic, Hervé Puy, Ingrid Winship, Jean-Charles Deybach, Laurent Gouya, Giel van Dooren, et al. 2015. "Red Cells from Ferrochelatase-Deficient Erythropoietic Protoporphyria Patients Are Resistant to Growth of Malarial Parasites." Blood 125 (3): 534–41.

Sonawane, Abhijeet R., Scott T. Weiss, Kimberly Glass, and Amitabh Sharma. 2019. "Network Medicine in the Age of Biomedical Big Data." Frontiers in Genetics 10 (April): 294.

Sundar, Shyam, and Bhawana Singh. 2018. "Emerging Therapeutic Targets for Treatment of Leishmaniasis." Expert Opinion on Therapeutic Targets 22 (6): 467–86.

Varikuti, Sanjay, Bijay Kumar Jha, Greta Volpedo, Nathan M. Ryan, Gregory Halsey, Omar M. Hamza, Bradford S. McGwire, and Abhay R. Satoskar. 2018. "Host-Directed Drug Therapies for Neglected Tropical Diseases Caused by Protozoan Parasites." Frontiers in Microbiology 9 (November): 2655.

Veras, Patricia Sampaio Tavares, Pablo Ivan Pereira Ramos, and Juliana Perrone Bezerra de Menezes. 2018. "In Search of Biomarkers for Pathogenesis and Control of Leishmaniasis by Global Analyses of -Infected Macrophages." Frontiers in Cellular and Infection Microbiology 8 (September): 326.

Walhout, Albertha J. M. 2011. "Gene-Centered Regulatory Network Mapping." Methods in Cell Biology 106: 271–88.

Weinkopff, Tiffany, Christoph Konradt, David A. Christian, Dennis E. Discher, Christopher A. Hunter, and Phillip Scott. 2016. "Leishmania Major Infection-Induced VEGF-A/VEGFR-2 Signaling Promotes Lymphangiogenesis That Controls Disease." Journal of Immunology 197 (5): 1823–31.

Weinkopff, Tiffany, Hayden Roys, Anne Bowlin, and Phillip Scott. 2019. "Infection Induces Macrophage Vascular Endothelial Growth Factor A Production in an ARNT/HIF-Dependent Manner." Infection and Immunity 87 (11). https://doi.org/10.1128/IAI.00088-19.

Wetzel, Dawn M., Diane McMahon-Pratt, and Anthony J. Koleske. 2012. "The Abl and Arg Kinases Mediate Distinct Modes of Phagocytosis and Are Required for Maximal Leishmania Infection." Molecular and Cellular Biology 32 (15): 3176–86.

Wishart, David S., Yannick D. Feunang, An C. Guo, Elvis J. Lo, Ana Marcu, Jason R. Grant, Tanvir Sajed, et al. 2018. "DrugBank 5.0: A Major Update to the DrugBank Database for 2018." Nucleic Acids Research 46 (D1): D1074–82.

Ylisastigui, Loyda, Nancie M. Archin, Ginger Lehrman, Ronald J. Bosch, and David M. Margolis. 2004. “Coaxing HIV-1 from Resting CD4 T Cells: Histone Deacetylase Inhibition Allows Latent Viral Expression.” AIDS 18 (8): 1101–8.

Zumla, Alimuddin, David S. Hui, Esam I. Azhar, Ziad A. Memish, and Markus Maeurer. 2020. “Reducing Mortality from 2019-nCoV: Host-Directed Therapies Should Be an Option.” The Lancet 395 (10224): e35–36.

Zumla, Alimuddin, Martin Rao, Ernest Dodoo, and Markus Maeurer. 2016. “Potential of Immunomodulatory Agents as Adjunct Host-Directed Therapies for Multidrug-Resistant Tuberculosis.” BMC Medicine 14 (June): 89.

Zumla, Alimuddin, Martin Rao, Robert S. Wallis, Stefan H. E. Kaufmann, Roxana Rustomjee, Peter Mwaba, Cris Vilaplana, et al. 2016. “Host-Directed Therapies for Infectious Diseases: Current Status, Recent Progress, and Future Prospects.” The Lancet Infectious Diseases 16 (4): e47–63.

6. Supplementary Files:

- Supplementary Figures:

https://docs.google.com/document/d/1TGsc5aqmrxjC6RnibZv8CvRQsEatAjcHcgYjRMfr4PY/edit?usp=sharing

- Supplementary Dataset 1:

https://docs.google.com/spreadsheets/d/16GVTcq1FOGC6cz1lz1OIXPU9Kozdg0v6F-0fQW5OCkI/edit?usp=sharing

- Supplementary Dataset 2:

https://docs.google.com/spreadsheets/d/19SV9SuSyy0w6zy42pdqd8Q3InVT6iLpzlm6KUnLJkrI/edit?usp=sharing

- Supplementary Dataset 3:

https://docs.google.com/spreadsheets/d/1xykTtLGj5ljBAg_DLOp1mbGf7_gsDzdzdnMOAlYT_Z8/edit?usp=sharing

- Supplementary Dataset 4:

https://docs.google.com/spreadsheets/d/1E_cyIbXBG_BQqA6ZJTClXigyVIVlonm9R-6EwwhAfNE/edit?usp=sharing

List of Figures:

Figure 1. Pipeline to identify potential therapeutic target for *Leishmaniasis* host-directed treatment in human macrophage from *RNA-seq* data.

Figure 2. Global transcriptomic profiles of *Leishmania* infected human macrophages and genes related to immune response and host-pathogen interaction. Distribution of *DEGs* between different specific time post-infection. The box width indicates the number of *DEGs* downregulated (blue) and upregulated (red) at adjusted *p-value* 0.05 and -0.5 > *FC* > 0.5. The color shading indicates *DEGs* that are involved in host-pathogen interaction and immune response according to *GO* functional analysis. The numbers at the end of each bar correspond to total *DEGs* obtained after paired-samples analysis.

Figure 3. Network comparison of non infected against infected macrophage at 4 hours post infection. A. The network shown is formed by 942 nodes (167 *TFs*) and 3847 edges colored according to their existence in the non-infected network, infected network or in both networks. **B.** Subnetwork represents all edges presented only in the 4 *hpi* network. Colors of edges and nodes are the same as the upper network.

Figure 4. Workflow Diagram to identify the Drug Targets using the Multipath package.

List of Tables:

Curriculum Vitae

Zaynab Hammoud

Education

2016 - 2021	**PhD student at the Georg-August-University Göttingen** PhD thesis “Using an n-layer graph model for representing and transforming knowledge on biological pathways” *Göttingen, Germany*
2015 - 2016	**Master 2 (Bac+5) in Computer Science and Telecommunication, Major: Information Research, Databases and Multimedia** Faculty of Sciences I, Lebanese University and Université Toulouse III, Paul Sabatier *Beirut, Lebanon and Toulouse, France*
2014 - 2015	**Master 1 (Bac+4) in Computer Science at the Lebanese University** Faculty of Sciences I, Hadath *Beirut, Lebanon*
2011 - 2014	**Bachelor in Computer Science at the Lebanese University** Faculty of Sciences I, Hadath *Beirut, Lebanon*

Professional Experience

2018 - present	**Research Assistant at the University of Augsburg** IT-Infrastructure for Translational Medical Research *Augsburg, Germany*
2016 - 2018	**Research Assistant at the University Medical Center Göttingen** Department of Medical Statistics *Göttingen, Germany*
March - September 2016	**Intern at the University Medical Center Göttingen** Department of Bioinformatics *Göttingen, Germany*

Publications

Auer F, **Hammoud Z**, Ishkin A, Pratt D, Ideker T, Kramer F. ndexr-an R package to interface with the network data exchange. *Bioinformatics*. 2018 15;34(4):716–7.

Hammoud Z, Kramer F. mully: An R Package to Create, Modify and Visualize Multilayered Graphs. *Genes*. 2018 Nov;9(11):519.

Hammoud Z, Kramer F. Multilayer networks: aspects, implementations, and application in biomedicine. *Big Data Analytics*. 2020 Jul 6;5(1):2.

Hammoud Z, Kramer F. Multipath: An R Package to Generate Integrated Reproducible Pathway Models. *Biology*. 2020 Dec;9(12):483.