UNIVERSITAT DE BARCELONA

UNIVERSITY OF CAMBRIDGE

TREBALL DE FI DE GRAU

TFG TITLE: Development of a GUI for InterMineR and Cytoscape to make biological databases FAIR.

DEGREE: Biomedical Engineering Degree

AUTHOR: Celia Sánchez Laorden

ADVISORS: Dr Rachel Lyne
Dr Gos Micklem

SUPERVISOR: Dr Santiago Marco

DATE: June 13, 2021

Títol: Creació d'una interfície gràfica d'usuari entre InterMineR i Cytoscape per fer que les bases de dades siguin FAIR.
Autor: Celia Sánchez Laorden

Directors: Dr Rachel Lyne
Dr Gos Micklem

Supervisor: Dr Santiago Marco

Data: 13 de juny de 2021

Resum

InterMine és un sistema del tipus magatzem de dades que permet crear grans bases de dades biològiques fàcilment des de fonts heterogènies en serveis web RESTful. El sistema proporciona eines estadístiques i d'anàlisis de dades potents. InterMine com a plataforma té una interfície web des d'on l'usuari pot fer cerques i incorpora recursos especialitzats com mètodes d'anàlisis d'enriquiment de conjunts de gens.

InterMineR és una de les biblioteques informàtiques que dóna suport a la plataforma des de l'entorn de programació de R Studio. En aquest projecte, s'han implementat noves funcionalitats per permetre que l'usuari treballi amb col·leccions i conjunts de dades creats des de el servei web, anomenats llistes. Les noves funcions han estat incorporades i publicades en la nova versió del paquet a Bioconductor.

L'objectiu principal d'aquest projecte ha estat la creació d'una interfície gràfica d'usuari (GUI) per fer cerques a les bases de dades d'InterMine i generar visualitzacions de xarxes Cytoscape. S'ha desenvolupat un programari executable en entorns d'escriptori. Aquest permet als usuaris reduir la complexitat de les dades i extreure'n significat amb fins de recerca. S'espera que permeti analitzar amb més profunditat les fonts d'InterMine i augmentar-ne la FAIR (dades trobables, accessibles, interoperables i reutilitzables en anglès).

Title : Development of a graphical user interface for InterMineR and Cytoscape to make biological databases FAIR.
Author: Celia Sánchez Laorden

Advisors: Dr Rachel Lyne
Dr Gos Micklem

Supervisor: Dr Santiago Marco

Date: June 13, 2021

Overview

InterMine is a biological data warehousing system for creating large biological databases of heterogenous data sources easily in RESTful web services. It provides powerful statistical and analysis tools. InterMine as a platform has a web interface in which user can run searches and incorporates specialized resources such as enrichment statistics.

InterMineR is one of the software 'client' libraries that interfaces the biological databases built using the InterMine platform with a programming environment, in this case R. In this project, new functionalities have been implemented to enable the user to work with stored collections of data and saved results sets created from a web-service, called lists. The new functions have been merged and published in the new Bioconductor version of the package.

The main aim of this project is to create a graphical user interface (GUI) to query the InterMine databases and generating Cytoscape network visualizations. We developed a software that runs on desktop environment. It allows users to reduce the complexity of the data and extract meaning from it. It will allow data in InterMines to be further analysed - increasing its FAIRness.

First of all, I would like to show appreciation to Dr Gos Micklem, for giving me the opportunity to conduct this project in his Lab.

I would like to express my most sincere thanks of gratitude to Dr Rachel Lyne for her guidance, advice and assistance. This work would not have been possible without her intense commitment and dedication.

I cannot fail in my remarks to express appreciation for the coworking with Yo Yehudi, Daniela Butano and Adrián Bazaga.

Finally, I would also like to thank to Dr Santiago Marco, for his advice and support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PROJECT ORIGIN AND MOTIVATION

This thesis was developed during an internship carried out the summer 2020 at the Department of Genetics of the University of Cambridge. The work done here is regrouped in a central project called InterMine, a data warehouse system, and has been supervised by Senior Biologist Rachel Lyne. From September 2020 to April 2021, this project has been extended and written.

During the last decade, a plethora of tools have been developed to explore research data from large biological databases. In particular, the development of software 'client' libraries in common programming languages to access the data is a must. The R programming stands out above the rest for its powerful statistical and graphical capabilities in the field of data science. The 'client' library InterMineR provides access to the InterMine databases through webservices and makes it possible to run complex data mining searches.

Having access to large databases is as important as having the tools to analyse them, as they are complex and heterogeneous. In particular, when complex interactions are considered, the creation of a network of multiple pathways of interest can bring out underlying biology. In this regards, InterMine team considers Cytoscape a powerful visualization tool that would increase the interoperability of the biological databases. Thus, giving to the users the resources to make their research data more Findable, Accessible, Interoperable and Reusable (following the FAIR principles).

In this project, it has been wanted to cooperate with the evolution of biological data integration and management. During the project, the InterMineR R package has been extended to enable the users access data directly from their computers without using the InterMine web application and communicating with the server through the R Studio interface. And, finally, it has been created a graphical user interface to querying the data of any of the InterMine databases and show the results in Cytoscape networks. All the new functions and tools have been properly documented with the purpose of making them more user-friendly.

Open source software promotes the development of powerful tools which we are reliant on, in our day-to-day. The work done here wants to be a little contribution back to the open-source community.

CHAPTER 1. INTRODUCTION

1.1. Scope

The main objective of this project is to contribute to the improvement of the InterMine platform[1]. InterMine is a widely used large-scale data integration platform for biological data. It has a web interface but also programming interfaces for all the main scripting computer languages (Python, R, Perl, JavaScript, Ruby or Java). This project is focussed on the InterMineR package and Cytoscape.

InterMine was created in 2002 at the University of Cambridge, originally as a Drosophila-dedicated resource, before expanding to become organism-agnostic. This has enabled the creation of many InterMines such as FlyMine, HumanMine and even a CovidMine. The InterMine team is part of the Micklem lab, headed by Professor Gos Micklem.

All InterMine code is freely available under the open source LGPL 2.1 license. All the information given when using the platform is treated in accordance with its privacy statement that can be read it in its website.

The following specific objectives have been identified in order to reach the first aim:

1. To improve the core InterMineR package.
2. To improve the integration of Cytoscape with R within InterMine.
3. To provide documentation and tutorials.

Other aspects such as improvements in the data browser tool, in client libraries or other visualization tools like MineViewer or Bluegenes for the improvement of the InterMine platform are not going to be consider in this project. InterMine is a big open source project and has many contributors working on it at the same time. In its GitHub can be found many repositories that need contribution in different issues.

The execution of this project is going to be carried out during the summer break under the remotely supervision of Dr Rachel Lyne, Ms Yo Yehudi, Ms Daniela Butano and Mr Adrian Rodríguez, from the Genetics Department of the University of Cambridge via email, Discord and Zoom. The internship entails a dedication of 36.5 hours per week for 8 weeks. After the final version of the software, some time will be devoted to refine and document all the work for the end-of-degree project presentation. The posterior drafting and finally the presentation of the end-of-degree project will be supervised by Dr Santiago Marco from the University of Barcelona.

1.2. Span

It is expected to improve the core InterMineR package and interface it with Cytoscape.

- Improvements of the core InterMine R package: Including missing functions that would make it easier for users to work with lists: getLists, newList, renameList and deleteList, initially. Lists objects are widely use in InterMine providing to the user information about a large set of bio-entities and facilitates powerful statistical analysis such as the gene set enrichment.

- Interfacing with Cytoscape: Cytoscape is a popular open source tool used to visualise biological data as a network or graph. Interfacing between InterMineR and Cytoscape software will involve running queries in R against data in InterMine databases and then “sending” it to the Cytoscape for visualisation. To make this easier for users an R shiny interface will be created.

Once finished, the data will be explored with the included applications coming up with some use-cases. User guideline for the interfacing between InterMineR and Cytoscape will be created.

1.2.1. Description of the span of the project

It is expected that the result incorporates all the necessary functions to the core of the InterMineR to make it easier for users to work with list objects. And so, if any other function appears to be useful, they will be incorporated extending InterMineR to provide the same functionality that InterMine has for the other languages such as Python.

Lists store collections of data and saved result sets from a web service. They are created from a list of identifiers or from a query result. This result can be obtained from a template query, a pre-defined search, or a personalize search made by the user using the InterMine’s custom query builder.

Cytoscape is available as a stand-alone program, as a web version and as an R package. It will be desirable to create a R shiny user-friendly graphical interface for users to run queries with InterMineR and then visualise the results with Cytoscape in node/edge graph-based visualisations and obtain networks of proteins or genes, for example. This interface will be complemented with use-cases.

1.2.2. Requirements of the project

To accomplish the objective of interfacing Cytoscape and InterMineR, the resulting work must feature certain functionalities. These basic requirements of the functions and interface must carefully be considered during the design process.

The requirements can be summarized in the following points:

- Mines compatibility: The user can access all the Mines available in a single user interface. This way, the user can explore a broad range of organisms and life science research areas by moving between databases.

- Query Builder: The user should have the possibility to choose a template query from a list of all the templates for an organism or mine. And on the other hand, the user should be able to build the query itself.

- Interactive and intuitive visualizations: Tools to facilitate the visualization and comprehension of the results are provided. The resulting Cytoscape networks can be edited by the user. Guidance edition tools are also desired to make the edition task easier.
- Dynamic and reactive user interface: The user interface changes dynamically in response to changes in the input controls, made by the user.
- Versatility: The application must be designed to be executed in any computer by any user with few installation requirements. Only desktop environments are expected to run this application; no mobile device is considered.

The following are the requirements of the project that have not been mentioned previously:

- All the code written will be documented and modularised.

The following requirements are for the end-of-degree project:

- Drafting and submission of the written project.
- Presentation of the work to the Court of the University. The final work must be well presented to the audience.

1.2.3. Limitations of the project

	Included	Excluded
Indispensable	– Extend InterMineR with functions to work with lists: *getLists*, *newList*, *renameList* and *deleteList*. – Interface InterMineR and Cytoscape to build up node/edge graph-based visualisations. – Document the written code. – Create tutorials for the Cytoscape functionality.	– Fixing the error of getModel function in FlyMine or HumanMine. – Perform list operations: difference, subtract, intersect, etc. – Fetch a list by name function.
Desirable	– Extend InterMineR core with other functions. – R Shiny interface for the interfacing between InterMineR and Cytoscape. – Provide use-cases in the tutorials.	– In the function *runQuery()* a parameter like "return.no.matches" that gives you the non-mapped names.

Table 1.1: Limitations

1.2.4. Deliverables of the project

Regular Zoom calls will be set up with the Micklem's lab, weekly initially or more often if needed to discuss the work. Once each section of the first work is completed, Dr Rachel Lyne and Ms Yo Yehudi (from Micklem's lab) will review it and merge it into the master project following an incremental, but linear, build model. For code review GitHub will be used. For the building of the Shiny interface the Agile methodology will be followed. This is a combination of iterative and incremental work sequences with focus on adaptability and client satisfaction. It begins with planning and continues through iterative development cycles that involve continuous user feedback and the incremental addition of features.

InterMine has a server in Discord with different channels or chats. There is a general channel, where public discussions are taken between all the members, and a support channel, where any developer can post a doubt and the others will help. For this project, a private channel will be opened with the supervisors.

1.2.5. Criterion of acceptation of the results

The main criterion is the gain of the functionality described in the span of this project. Once each contribution, in the form of a pull request, in a section has been reviewed by the maintainers (Yo, Daniela and Rachel), it will be added or merged to the IntermineR repository master branch.

Other criteria for acceptance (that could be used) are:

- Passing unit test for new code (if applicable).
- Passes all tests – according to Travis CI.
- Documentation (if applicable).
- Detailed commit messages.
- Well commented code.
- Checkstyle.

1.2.6. Restrictions of the project

This project was initially plan as an internship abroad but due to COVID-19 it will be done remotely. This modality of work can be a factor of risk for correct execution of the project. For that reason, it will be essential to control and reinforce the communication.

The second restriction will be the time for executing the project. Since part of the 8 supervised weeks will be devoted to document (myself) before starting programming as the developer (I) does not have previous experience of all the involved tasks.

1.2.7. Initial known risks

Having little experience working with InterMine queries could delay the anticipated completion of this part. Moreover, having never worked with Cytoscape leads to a learning process to be able to merge it with InterMineR.

The COVID-19 situation makes compulsory teleworking. For the supervision of the work and the correctness of it, not being physically present is a risk as the communication will be less fluent and problem-solving is assumed to be slower.

Finally, working beyond the deadline of the internship will not be desirable.

CHAPTER 2. BACKGROUND

2.1. State of the art

2.1.1. InterMine as a Data Warehouse

A data warehouse is a storage area for collecting data which may have been gathered from a source or multiple sources via an integration layer that transforms data to meet the criteria of the warehouse. In a centralized data warehouse such as InterMine, since data is unified and in homogeneous format, queries are easier to write and gives user better performance than accessing multiple distributed data warehouses.[2]

Many biological data management systems or data warehouses have been developed to integrate huge amounts of this data in one place. Each of them is appropriate for different scenariosAnd some examples are BioMart, The Eukaryotic Pathogen Databases (EuPathDB) and BioCyc.[3]

A short summary of the main biological warehouses is given below, they are presented and described in chronological order of realisation.

One of the oldest found data warehouses is **BioCyc**[4], released in 1997. It is a collection of more than 3000 organism-specific Pathway/Genome databases (PGDBs), and is particularly focussed on microbes. Among the tools it offers there are Omics Viewers. Three years later appeared **GIMS**[5] to support the analysis of genomic data with the construction of canned queries that can be reused. In 2004, the **ONDEX**[6] framework was registered. Currently, it offers an environment for text mining, large scale database integration and graph analysis highly functional tools for genes, genomes, and proteins. A strength is that all data that are relevant for a given analysis is extracted from any combination of integrated databases and text sources and no a priori knowledge and pre-selection of databases is required. Almost simultaneously, **GUS**[7], **Biozon**[8], **BioWarehouse**[9] and **Atlas**[10] appeared. GUS was specialized in functional genomics but could be generalized to the rest of omics and clinical records. Biozon unified DNA sequences, proteins, interactions, and cellular pathways and was able to store functional predictions. BioWarehouse approach was more about general bioinformatics research, enabling multi-database queries and data mining of relevant data sets. Atlas, based on relational data models and very similar to BioWarehouse, integrated biological sequences, molecular interactions, homology information, functional annotations of genes, and biological ontologies. In those days, **BioMart**[11], and one later, **EuPathDB**[12] were realised. The first one was an effort to integrate over 800 biomedical databases of genomics, proteomics, model organisms, cancer data ontology and more. While the second was specialized in genomic and postgenomic data from eukaryotic pathogens along with non-pathogenic species and select pathogen hosts. Both query strategies were based on predefined searches or filters. And again, both support many third packages for visualization purposes such as Cytoscape. Finally, it can be remarked two more data warehouses that appeared in 2009 and 2012, respectively, **BioXRT** and **OGeR**[13]. BioXRT does not support data mining but offers an easy path for scientist to develop their own databases. OGeR was restricted to prokaryotic genome data although it does not provide tools for clustering and statistical analysis nor does it provide advance mining tools. OGeR and BioXRT are no longer available. Same happens with GIMS, Atlas, GUS and Biozon. BioWarehouse was shut down in 2015 but the source code is still available for download. BioMart, EuPathDB, BioCyc and Ondex are still updated in 2020.

While Ondex version must be downloaded and requires a Java run time environment, BioMart, EuPathDB and BioCyc offer a web server. Ondex has a ToolKit user interface for visualization of networks, the ONDEX frontend, and another for data integration and data mining, ONDEX backend. The second is only advisable for advanced users and developers.

BioMart offers programmatic access through Perl and Java API's, RESTful web services and SPARQL, and as mentioned before its web interface. It is also a cross-platform that supports many third-party packages such as Galaxy, Cytoscape and biomaRt, which part of the Bioconductor library.

EuPathDB includes 13 web pages following the same web site structure. This offers rich data mining capacity supported by the Genome Browser and a private Galaxy workspace for visualization and primary data analyses, respectively.

BioCyc website offers querying and analysis tools such as omics data analysis, metabolic map diagrams and models, pathway collages, regulatory network diagrams, comparative analysis, and BLAST search. In addition, a downloadable software/database bundle includes functionality not available in the web server and executes faster. The API allows to query via Java, Perl, and Common Lisp languages.

In the Scan of the Market section is going to be compared BioMart, EuPathDB and BioCyc with InterMine.

2.1.2. Setting up an InterMine: making data FAIR

Making data FAIR means making data **F**indable, **A**ccessible, **I**nteroperable and **R**e-usable. The principles of FAIR were first formally published in Scientific Data in 2016[14]. InterMine currently has funding to increase it's conformance to the FAIR data principles, by the BBSRC Council.

Currently, the accessibility to the data is guaranteed through web apps, web services and client libraries for the most common languages and by a very sophisticated query system. The usability and interoperability is provided by the exportation of the results in different formats. And reproducibility is also ensured by the automatic code generation function that reproduces the code in different languages of a concrete specified query. Other way to access the data is through visualization tools. Along these lines, Blue Genes GUI is designed to make searching and analysing genomic data easy. In this way, the project of interfacing InterMineR and Cytoscape aims to be a modest contribution to making InterMine FAIRer. Contributing to the interoperation and reusable aspects of FAIR, by allowing two software components to interoperate and scripts to be saved allowing researchers to easily re-create analysis workflows.

Aspects aimed at improving the FAIRness of InterMine include permanent access

URLs, mark-up of web pages for machine findability and improvements in ontology's usage and license information. These improvements have been implemented with success. However not all data within InterMine has a data license and efforts will continue in this area. Another project aims to help transform research data files into cloud hosted InterMine databases. InterMine Cloud project attempts to lower the barrier of setting up an InterMine instance for researchers with little programming knowledge. As a part of InterMine Cloud, InterMine Boot aims to allow user to setup locally InterMine instances.

From the data warehouses mentioned in the previous section, EupathDB has associated a BRC project names Eukaryotic Pathoge, Vector and Host Informatics Resource (VEuPathDB) that declares to follow FAIR principles. It is limited to eukaryotic pathogens and invertebrate vector of infectious diseases, including data from prior projects of parasitic species (EupathDB), fungi (FungiDB) and vector species (VectorBase). VEuPathDB helps scientist to submit genomic-scale data related with the mentioned fields. Looking at the databases in FAIRsharing, a platform that ensures that databases among others are aligned with FAIR data principles, five BioCyc's are found.

2.1.3. Cytoscape: graph analysis software

The growth in high-throughput genomics, transcriptomic and proteomic techniques has led to an explosion in large biological data sets. This provides a challenge in how best to analyse and visualise these data. Networks provide a good way to visualise such data, enable patterns and relationships in the data that can be explored. Focusing on network visualization tools that are free, open-source and have developer packages available in programming languages such as R, five are presented below.

Gephi[15] is highly interactive, and users can easily edit the node/edge shapes and colours to reveal hidden patterns. It assists users in pattern discovery and hypothesis making through efficient dynamic filtering and iterative visualization routines. It stands up in terms of scalability and memory efficiency, being a great tool for layouting large-scale network (nodes 10^4, edges 10^6). Also, for large-scale network visualization, **Tulip**[16] is one of the easiest-to-use tools due to its simplicity and guided interface. It offers enabling edge-bundling algorithm. For massive networks with more than 10 billion nodes, **Pajek**[17] is the best scalable tool. However, it lacks operating system interoperability and input file format flexibility and not good visualization features compared with others. Not all offer interactive user experience, this is the case of **GraphViz**[18] which runs from the command line. It focusses on 2D graph layout algorithms to provide a static aesthetically pleasing view of the network.

It is not easy to compare these tools with each other as they serve different purposes. Nonetheless, **Cytoscape**[19] is the most preferred tool for biological and biomedical analyses, as illustrated in the statistics of publications for the tool[20]. It is accompanied by more than 200 plugins, which are additional features available as Apps. Compared with the rest, it has the richest palette of predefined colour styles, the most efficient collection of clustering algorithms (layouts), and the best network profiler for intranetwork comparison of topological features as indicated by the large number of citations it recieves. Moreover, it is an extensible software, meaning that it can be grown and implement an extension[21]. Indeed, the most used extension is the Cytoscape.js JavaScript library[22]. Cytoscape.js is an API and the successor of Cytoscape Web. It is designed to be a building block for complex data visualization web applications. Its interoperability is relevant when combined with network data generated by igraph, which can be programmed in R, Python, Mathematica, and C/C++.

2.2. Project Environment

This project aims to be a continuity of the work carried out by Bing Wang and then modified and extended by Konstantinos Kyritsis on the InterMineR package. There are several additions that could still be made to the package to cover all the functionality currently available through the webapp. These improvements are geared to facilitate working with lists.

In addition, there is the opportunity to integrate the graph-drawing package, Cytoscape, with R Shiny and InterMine. As mentioned before, tools to set analysis and visualizations directly available from an InterMine increases its FAIRness.

CHAPTER 3. SCAN OF THE MARKET

As stated, biological data warehouses are a powerful research tool in bioinformatics and biotechnology. In bioinformatics, data warehousing can help biologists to select and design critical experiments. In biotechnology, transformation of data to knowledge also called knowledge discovery from databases (KDD) is a goal for users. KDD can be defined as the "non-trivial extraction of implicit, previously unknown and potential useful information from data"[23]. These are only some of the applications of datawarehousing in biology research. And as applications evolve, the data warehouses available in the market do so as well. As it has been summarized in the State-of-the-Art section, currently, the more well-known are BioMart, The Eukaryotic Pathogen Databases (EuPathDB) and BioCyc.

The future work of those is heading towards promoting the collaboration and code reuse in a context of open-source software. One of the specific goals is to make users participate actively in the process of building the databases for the data warehouse and indirectly expand the data further over the coming years. Moreover, graphic visualizations are a common working aspect for future releases.

To explore the position of InterMine, and detect its hallmarks, a comparison between BioMart, EuPathDB, BioCyc and InterMine is presented in the appendix D.1.

For BioMart community portal as it is temporarily unavailable, further information is provided from one of the BioMart community servers, Ensembl[24]. The list of those servers is listed in its website.

Following with BioMart, the Cytoscape Core App provides support for BioMart web service in Cytoscape.

From Table D.1 in appendix can be concluded that each data warehouse has its sector in which stands out. The biggest data warehouse in number of species is BioCyc followed by EuPathDB. Both also offer good analysis tools. However, InterMine is the unique offering the possibility to customize from scratch a search or a query and automatically give the code of this query in many programming languages. Moreover, InterMine is offering API's in more programming languages and developing an R package that will improve the visualization of the results with Cytoscape. Currently, it is also developing other tools for visualization. Finally, InterMine aims to provide the tools to the clients for developing their own InterMine app and so expand the platform. For this reason, it is also offering a lot of tutorials and paths to customize the InterMine apps and websites.

Several applications designed for network analysis and the creation of network graphs such as gephi and graphviz exist. Although not specifically designed for it, R can be powerful tool for the same purpose. Comparing R with the isolated network analysis software's that have been described previously, R presents a clear advantage. Firstly, R enables reproducible research not possible with the other applications. Secondly, R represents a powerful data analysis tool to manipulate data and prepare it for network analysis. And finally, packages such as igraph, gggraph, graphlayouts or snahelper are growing making even more complete R network analysis. The features and functionality of the stand-alone software's described are already available through R libraries. For example, this is the case of rgexf package for Gephi or DiagrammeR for Graphviz. However, the most completed and diverse packages are the ones devoted to enhancing the interoperability of Cytoscape and R[25][26]. Finally, the combination of Shiny apps with packages for Cytoscape Interactive Graph Visualizations is being explored by data scientists for no more than two years, although the building of the R packages comes further in time.

CHAPTER 4. DESIGN ENGINEERING

4.1. Improvements of the core InterMineR R package

4.1.1. Preliminary Project Study

In this part, it is expected to improve the functionalities of the InterMineR package to work with lists. This has been done for the InterMine Java, Perl, Python and JavaScript packages. The InterMine functionality is exposed over an HTTP API (RESTful) and the Client Libraries, or packages, are close to mirroring the HTTP API features.

As regards List functions, the documentation of the code for the packages includes:

- Java: It has the `org.intermine.client.lists` Package and inside this one the Class `Lists` and `ItemList`. The Methods for the two Class are documented with a Description (in the Java API). However, the code for each is not visible.
- Perl: It has the `Webservice::InterMine` cookbook which is a set of short tutorial 'recipes' that aim to demonstrate particular features of the `Webservice::InterMine` Perl API. Each recipe presents some code followed by a section which explains and discusses the features used. `::List`, `::List::Upload recipe`, `::List::Enrichment` recipe and `::List::Combination` recipe are devoted to lists. These tutorials are very useful for users however if we want to read the code behind these functions, we need to go to the *intermine-ws-perl* InterMine GitHub repository and into *List.pm*.
- Python: It has the `intermine.lists` package accessible through the Python API. The functions are displayed in modules where a description of the functionality is given for each. In the *intermine-ws-python-docs* GitHub are 12 tutorials in Jupyter format. Tutorial 9, 10 and 11 explored different functions of the `intermine.lists` package. However, if we want to see the code behind this package, we need to check the `intermine-ws-python` InterMine GitHub repository, specifically the folder `lists` inside the folder `intermine`.
- JavaScript: In the JavaScript API, there is a section called `List` in which the different functions documentation is displayed. However, to see the code behind these functions we need to check the `imjs` InterMine GitHub repository, go to the `src` folder and open `lists.coffee`.

4.1.2. Proposed Solution

Among the different ways to proceed, it is chosen to explore how the InterMine Python package works with lists, specifically for the functions *getLists*, *newList*, *renameList* and *deleteList*. The main advantages of the `intermine.lists` package are the extension of it and the documentation. Moreover, Python is the only listed programming language known. The figure from appendix D.1 is a scheme of the

`ListManager` class. It is made to understand the dependencies on other functions that the four methods that want to be replicated in R Studio (in navy blue) present.

Additionally, I/O Docs of InterMine has helped to identify which are the paths passed to post and get http requests. It is a live interactive system for `RESTful` web APIs where documentation from different mines services is found.

The documentation of the new functions will be presented in the standard way of documenting the objects in a package; writing Rd. files in the man/directory.

4.2. Interfacing with Cytoscape

4.2.1. Preliminary Project Study

Option A: Read the RCytoscape and InterMineR packages documentation. Work on a R script to run a query and pass the result to the RCytoscape functions for visualization. Then, implement the workflow on a Shiny R interface.

Option B: Explore other packages from Cytoscape documentation. Work directly on a Shiny R interface. The interface should include the following workflow: run queries in R against data and visualise the results with the Cytoscape tools.

Option C: Biomart Web Service Client model. Cytoscape Apps Ladder. One of the requirements, and restrictions, is JDK (Java SE Development Kit) 11.

4.2.2. Proposed Solution

The option C is excluded due to the lack of previous experience working with Java. The option A is discarded although it was the proposed initially. RCytoscape functions use XML-RPC connection to communicate between R and Cytoscape. XML-RPC is a remote procedure call protocol that uses XML to encode the calls and HTTP for transport. However, the networks are only displayed in Cytoscape visual interface. So, the fact of not having the functions to display the visualizations in the same R Studio makes very difficult to work with Shiny R interface. The **option B** is presented as the more straightforward way to end up with a useful tool. Tasks to be done, before designing the interface, are broken down into these steps:

1. Read the Cytoscape documentation to know what offers.
2. Explore the template queries from a Mine (HumanMine for example) retrieved by the InterMineR functions.
3. Comprehension of the paths, the interconnection between data classes, types, and arguments among other things, reading the data tables with the results.
4. Search for existing packages that interface Cytoscape with Shiny.

4.3. Tutorials

4.3.1. Preliminary Project Study

Tutorials are an essential tool to get to know the platform. Under the InterMine platform we can find a great variety of tutorials and tools to understand better how all works.

FlyMine offers an extensive manual and videos under the 'help' link. These apply to all InterMine databases. There we can watch a brief overview of FlyMine, do 9 exercises and look for the solution, review a set of worked use-cases under the Cookbook section and read documentation of each function in the warehouse.

YeastMine and TargetMine offer videos demonstrating the main functionalities of each warehouse. The first one, YeastMine, videos are uploaded in its Youtube channel. There are 15 short videos of between 1 or 3 minutes. And the videos from TargetMine are a brief introduction for TargetMine system, an introductory movie for TargetMine and 4 basics of TargetMine (quick search, template queries, list functions and queryBuilder). These are uploaded in its tutorials page.

Finally, the intermine-python package tutorials are in the form of Jupyter-Notebooks. The last tutorial, Tutorial 14, is about Visualisation.

The different formats of tutorials are:

- Videos
- Exercises to solve + solution
- Use-cases
- Jupyter-Notebooks

4.3.2. Proposed Solution

For the tutorials of the InterMineR interface with Cytoscape, use-cases will be provided in short video format. These will be accessed through GitHub repository and the same Shiny interface.

4.4. Documentation

4.4.1. Preliminary Project Study

It is important to describe the functions that will be created. In this way, every function in InterMine platform is explained in a user guide. The user guides or documentation pages can be found in different formats depending on the language of the package, the InterMine organism, or simply the applications or purpose of the functions. It can be found information for even create your own InterMine, build a database or customize your web application, among others.

As with Tutorials, documentation can be displayed in GitHub repositories or in the help pages of the InterMines in HTML. But also, it can be found at MetaCPAN for the Perl Client Library or at Bioconductor for the R Client Library.

The documentation describing functions of Java Client Library has for each function a description of it and the accepting parameters. In some cases, they include what this function returns or other considerations.

Other documentation, such as for Python which can be found in GitHub and HTML or for Perl, includes some examples and displays the functions with a little description of each, no more than a sentence.

To document the interfacing of InterMineR and Cytoscape we have different options, these are some of them:

- GitHub repository.
- HTML through a page of InterMine.
- HTML though Bioconductor.
- R Script through Bioconductor.
- Document in the same R Shiny interface, creating a section for it.
- Submit a PDF document.

4.4.2. Proposed Solution

Considering the deadline and the lack of previous knowledge of HTML the documentation will be write first in LaTeX to have a user guide in a PDF format. Once this is done, if the Shiny R interface has been done the PDF document content will be included in it.

The minimum information that should contain this PDF is a brief introduction of the Shiny interface and the basic functionality achieved with the user interface. It will be desirable to include one example or demonstrative figure.

CHAPTER 5. DETAIL ENGINEERING

5.1. Technologies involved

The first task, the improvements in the core of InterMineR, required concrete and known environment, R Studio. The `httr` package is used to send data to the server or make a request and to get data send back from the sever, the response. Complementary, the `utils` package is used to percent-encode characters in URLs. The `S4` class in R is a system for object-oriented programming. It is implemented in the methods package. Finally, the NAMESPACE of the package is generated using `roxygen2` package. The documentation for the new methods and classes has been created.

Regarding the second task, having the opportunity of building an application from scratch also offers a wide array of choices. Choosing `Shiny` package to build the app has been made paying special attention to the requirements introduced before. Once the main environment is defined, additional tools are also added to the development such as HTML and multiple R packages that are grouped by utility and detailed as follows.

Functionality	Source	Libraries	Functions
Shiny interface - App structure: tabs, buttons, input controls, outputs, modals. . .	CRAN	– shiny – shinydashboards – shinycustomloader – shinyalert – shinyBS – shinyFeedback – rintrojs – shinyWidgets – htmltools	actionButton(), column(), conditionalPanel(), downloadButton(), downloadHandler(), eventReactive(), fileInput(), fluidPage(), fluidRow(), htmlOutput(), icon(), isolate(), mainPanel(), modalDialog(), observe(), observeEvent(), radioButtons(), reactive(), reactiveVal(), reactiveValues(), removeModal(), renderDataTable(), renderText(), req(), runApp(), selectInput(), showModal(), sidebarLayout(), sidebarPanel(), sliderInput(), submitButton(), tabPanel(), textAreaInput(), textInput(), uiOutput(), updateSelectInput(), updateSliderInput(), updateTabItems(), updateTextAreaInput() // box(), dashboardBody(), dashboardHeader(), dashboardPage(), dashboardSidebar(), dropdownMenu(), dropdownMenuOutput(), menuItem(), menuSubItem(), messageItem(), renderMenu(), sidebarMenu(), tabItem(), tabItems(), updateTabItems(), withLoader() // shinyalert(), useShinyalert() // bsModal(), bsTooltip() // hideFeedback(), showFeedbackDanger(), useShinyFeedback() // introjs(), introjsUI() // colorSelectorInput() // div(), tags(), br(), hr(), includeHTML(), includeMarkdown(), strong()
	GitHub	– shinyCheckboxTree	checkboxTreeInput(), updateCheckboxTreeInput()

Strings Modifications and information.	CRAN	– R.utils – base	printf(), decapitalize() // as.numeric(), duplicated(), replace(), paste(), unique(), identical(), nchar(), toupper(), tolower(), strsplit(), length(), substr(), is.null()
Lists and dataframes Modifications and information.	CRAN	– base – rlist – dplyr – data.table	list(), data.frame(), duplicated(), replace(), unique(), identical(), nchar(),length(), names(), is.null() // list.append() // select(), mutate() // subset(), rbind()
Display datatables	CRAN	– DT	DTOutput, datatable()
Network analysis - Including Cytoscape functionalities to visualize interaction networks.	CRAN	– igraph	igraph.to.graphNEL(), edge_attr(), vertex_attr(), graph_from_data_frame()
	GitHub	– cyjShiny – cytoscape	cyjShinyOutput(), renderCyjShiny(), doLayout(), getSelectedNodes(), fit(), fitSelected(), hideSelection(), invertSelection(), savePNGtofile(), renderCyjShiny(), selectFirstNeighbours(), selectNodes(), showAll() // cola_layout(), cytoscape(), cytoscapeOutput(), layout(), node_style(), panzoom(), renderCytoscape()
	Bioconductor	– graph – RCyjs – Rcy3	nodes() // clearSelection(), dataFramesToJSON(), sfn-method
Save images Saving the resulting networks	CRAN	– png – webshot – htmlwidgets	writePNG() // webshot() // saveWidget()
Write files and save zip folder	CRAN	– utils – base – base64encode – RJSONIO – zip	write.csv(), unzip(), read.csv(), fromJSON() // base64decode() // tempfile(), setwd(), writeBin(), close(), Sys.time(), write() // zip()
Query Integrate databases of any Mine and run data mining queries.	GitHub	– InterMineR	initInterMine(), listMines(), getModel(), getTemplates(), getTemplatesQuery(), summary(), setConstraints(), setQuery(), runQuery, simplifyResult()

Table 5.1: R packages.

Hypertext Markup Language (HTML) is the main language for creating web pages. It is a Markup language (not a programming language) and depends on JavaScript for executing processes. In Shiny depends on the R package `htmltools`. The function *includeHTML()* is used. An HTML document usually contains references to files containing CSS code and JavaScript code which act on the page. Cascading Style Sheets (CSS) is the style sheet language used for describing the presentation of HTML pages. CSS provides many exclusive features and a way to take control of the style of a web page separately from its content. CSS can also be nested in the same document as shown in the appendix B.1 with CSS embedded in `<style>`.

The code editor that has been used is Visual Studio Code (VS Code) for the first task and R Studio for the second task. VS Code is an open-source text editor developed by Microsoft. It includes debugging support, integrated Git control, syntax highlighting, smart code completion, snippets, and code refactoring. GitHub Desktop is a graphical interface that integrates the main features of Git making the development process much easier. The integration of VS Code to GitHub Desktop has let creating Pull Requests and thus contribute directly to the master branch. The app has been tested on Microsoft Edge.

5.2. Design

5.2.1. Improvements of the core of InterMineR

The results of a query run against InterMine dependencies returns a set of data that can be identified by what is called a primary identifier. It is useful to save them, together with the results, for further analysis and the best way is in a list in our InterMine account, so we can use it again in a later query. Having investigated on the Python `ListManager` class, it has been discovered that this class methods offer different possibilities to manage list contents and operations. The functionalities that have been replicated in the new version of InterMineR are get, delete, and create a list, search for an unused list name, and do operations with lists such as intersect, union, difference and subtract.

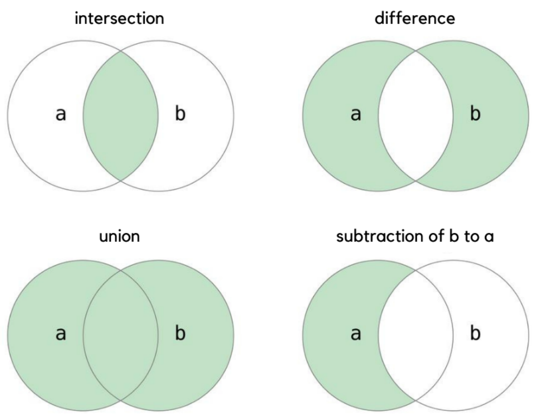


Figure 5.1: Operations with lists.

The following is a diagram of the HTTP request/response model that communicate the user with the InterMine webservice. It has been useful to identify the complementary functions that have to be built. From the status line one can know if the request has been successful, by looking at the http status code. The three-digit code of a successful request is `200`. To access the body of the request the *content()*

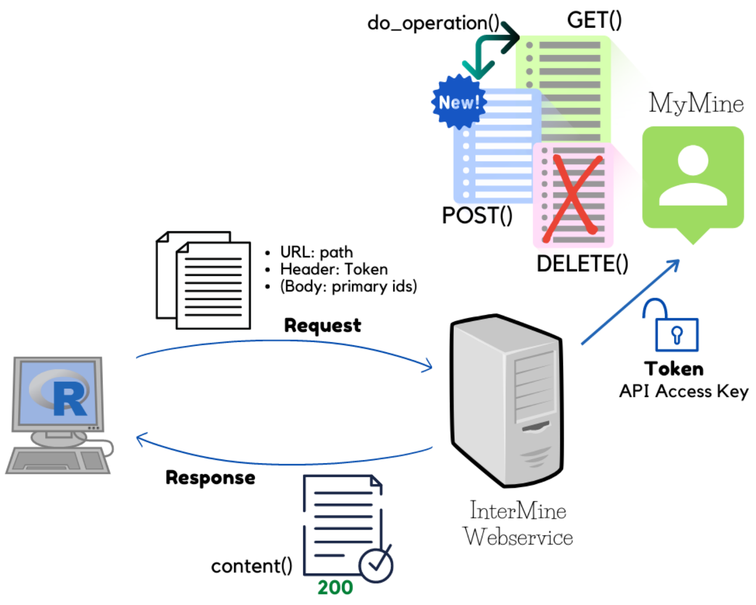


Figure 5.2: HTTP request/response model.

function can be used. Indicating that the encoding is `ISO-8859-1` and the desired type of output is parsed whom can access the lists parser to see the requested data. If, instead, data want to be sent to the server *POST()* has to be called. As been said, it is required to include the possibility of deleting list and for this *DELETE()* is called. Again, the three pieces compose the request when posting and deleting. Here, it is important to consider the header part to indicate the credentials to authenticate a user agent with the server, in this case the Token or API Access Key. The status line defines the URL where additional data can be sent to the server with the query string. Only when posting to create a list, the body contents are the primary identifiers list. In this case, the name, description, list type and organism are sent in the URL. When deleting, the name of the list to delete and the Token are sent in the URL. And when doing operations with lists, the corresponding path 5.1, the new name for the resulting lists, the lists to operate and the description is sent in the URL. Additionally, tags to categorize the new list can be specified at the end of the URL. The user can do intersect, union and difference with many lists as considered but when doing subtract the user must specify two kind of lists, source list and subtraction lists.

```
#' @rdname webservice-methods
#' @exportMethod list_manager
setGeneric("list_manager", function(object,...){
  standardGeneric("list_manager")
})
#' @rdname webservice-methods
#' @exportMethod list_manager
setMethod(
  "list_manager",
  signature(object = "Service"),
  function(object,...){
    return(new("ListManager",
               DEFAULT_LIST_NAME = 'my_list',
               DEFAULT_DESCRIPTION = 'List created with R client library',
               LIST_PATH = '/lists',
               INTERSECTION_PATH = '/lists/intersect/json',
               UNION_PATH = '/lists/union/json',
               DIFFERENCE_PATH = '/lists/diff/json',
               SUBTRACTION_PATH = '/lists/subtract/json',
               mine = object@mine,
               token = object@token))

})
```

Listing 5.1: webservice-methods.R list_manager

Having identified how the HTTP request/response must work the development of the functions is going to be described.

The `ListManager` class has been created in a R script with the same name. Inside representation, the list of slots or attributes are character vectors that are defined in the `list_manager` method (see appendix A.5). This method is from `Service` class (see appendix A.2) and initialize the slots of the `ListManager` class with the different paths that will be required in the URL to make a request (previous figure). It also contains the mine chosen by the user and the Token, which come from the `Service` class definition in *initInterMine()* function (see appendix A.4).

1. Service class: It is the main interface for the user. It will provide access to queries and templates, as well as doing the background task of fetching the data model and requesting the query results. Objects can be created using the function *initInterMine()*.

2. InterMineR class: It constitutes a class used to store the information which are required for performing a query for biological data in an InterMine instance. Specifically, it contains information about:

   (a) the type of data which are to be returned from the InterMine instance,

   (b) the type of sorting performed on these data, and

   (c) the constraints used to perform the query for the data of interest.

   Objects can be created using the function *setQuery()*.

3. ListManager class: It constitutes a class used to store the information required for managing lists contents and performing operations. Specifically, it contains information about:

   (a) the default list name and description,

   (b) the different URL endpoints, and

   (c) the information of the WebService.

   Objects can be created using the function *list_manager()*, which is a `webservice` method.

The methods for the `ListManager` class are defined in a R script with the same name. Three methods and one function have been auxiliary defined to be called inside the main methods which are *get_list*, *delete_list*, *create_list*, *intersect*, *union*, *difference* and *subtract*. The auxiliary methods are; *GET_api_list* which returns the response object of the Request, *get_unused_list_name* which checks if the name given by the user has been already used and, in such a case, provides a new one, and *do_operation* which creates a new list results of an operation. To see the detail of each function you can read the appendix A.6.

Correctly documenting the functions from a package is a key step to publish it. Users need it to know how to use the package but also is useful for the developer and future developers to extend it. The object documentation method is the chosen as can be accessed by `?` or *help()*. To do so, .Rd files are written in the man/directory in a concrete syntax. When you use `?` function, *help("function")*, or *example("function")*, R looks for an .Rd file containing `\alias{"function"}`. It then parses the file, converts it into HTML and displays it. These files use a custom syntax, loosely based on LaTeX. One of the Rd files created can be found in the appendix A.7. The main commands that have been used to write the new documentation have been:

– `\name{name}`, `\title{Title}`, `\description{...}`: the header provides useful information about the objects documented.

– `\examples{...}`: examples of how to use the function. Code in this section is set in typewriter font without reformatting and is run by *example()*.

– `\author{...}`: information about the author(s).

– `\seealso{...}`: allows you to point to other useful resources.

– `\section{...}`: information is given within a series of sections with standard names.

  – `\arguments{...}`: Description of the function's arguments, using an entry of the form: `\item{argument}{Description of argument}`.

– `\code{\link{...}}`: format inline code and link to other documentation, in this case to other functions from the package.

The package roxygen2 is used to generate Rd documentation, `NAMESPACE` file, and collation field using specially formatted comments. The main advantage it offers is keeping the documentation up-to-date as the documentation is written in-line with code. In this project roxygen comments have only been used to create the `NAMESPACE`. Roxygen comments start with `#'` and come before a function to distinguish them from regular comments, the collection of comments is called a block. Blocks gain additional structure using tags. `@rdname` tag is used to control where method documentation goes and document multiple functions in the same file. `@import` tag is used to import all the functions from a package and `@importFrom` to import a specific function from a package. If instead, what is wanted is to export an object `@export` tag is put. Specifically, when the object is a method it is used `@exportMethod`. The last tags commented enable roxygen automatically generate the right directive in the `NAMESPACE`. Once the tags are placed, just running *devtools::document()* (or pressing `Ctrl/Cmd + Shift + D` in RStudio) the roxygen comments are converted to .Rd files.

At the end of the task, when the results have been accepted by the supervisors Daniela and Yo, a Jupyter Notebook for R has been created replicating the one written with the Python API. The results obtained with it are going to be discussed in the Results section 5.3.

5.2.2. Interfacing InterMineR and Cytoscape with Shiny

The Shiny interface allows researchers, without software experience, to run queries and interpret them with Cytoscape networks without prior software experience. The tasks have been divided in seven tabs inside the app as the next workflow diagram 5.3 represents.

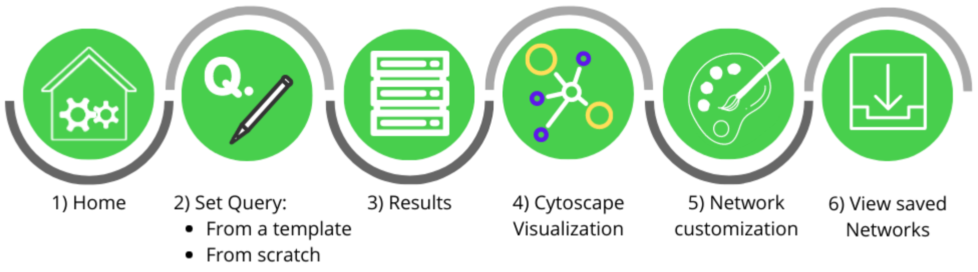


Figure 5.3: Workflow for the Shiny Interface.

The following describes the general architecture of the Shiny interface. Then, the design of each tab is going to be analysed individually.

The structure of a Shiny App is defined in the main file app.R, where the app is contained. The app.R file has three components: a user interface object called `ui`, a `server` function and a call to the `shinyApp` function (see B.2). The `ui` controls the structure of the app while the server function contains the instructions to build it. The `ui` is a dashboard page, a facility provided by `shinydashboard` package. This type of layout divides the screen in three sections; header, sidebar and body as can be seen in the scheme 5.4. Number 1 in the scheme corresponds to the dashboard header which includes a dropdown menu. The dropdown menu, number 2, gives access to the GitHub repository where the source code is published and to the Issues section of the repository through message items, message inside the menu. Moving to the dashboard sidebar includes a select input control, number 3, and a sidebar menu, number 4. The select input enables the user to select the InterMine to start the query. The list of InterMines displayed is obtained from the function *listMines()* from InterMineR package (see B.5). Once this is done, two more functions from this package are called. *initInterMine()* function saves the selection of the user in a list class (see B.5) and *get-Model()* the specifications of the model in a multilevel list (see B.5).

A *selectInput* is just an example of the numerous control widgets that Shiny provides. A widget control is a web element that the users can interact with and send messages to the Shiny app. They collect a value from the user, that when the widget is changed it changes as well. The first two arguments when including a widget are the name and the label. The name will not be seen by the user and enables the access of the widget's value, just by indicating `input$name`. The label, a string, is what appears in the web-app.

In the sidebar menu (number 4 in Figure 5.4), the seven tabs of the app are *menuItems* or *menuSubItems* and the user can go directly to a tab by clicking on its corresponding *menuItem*. Moving to the dashboard body or tab panel (see Figure 5.4)

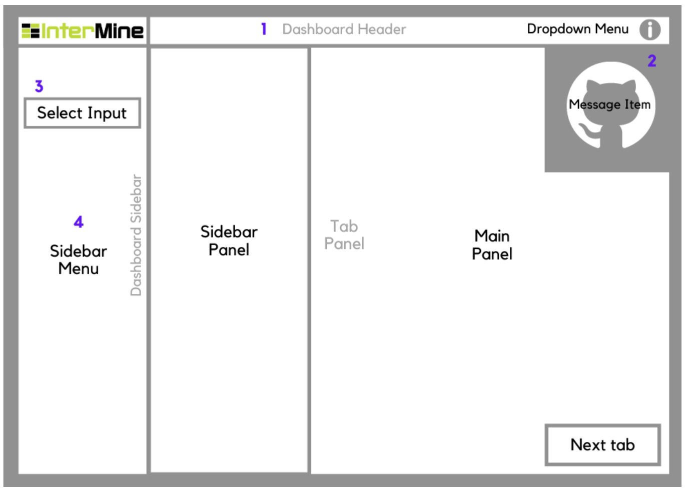


Figure 5.4: Layout divided into Header, Sidebar and Body.

two layouts configurations can be found depending on the tab.

For sub-tabs “Template” and “Query builder”, and tabs “Visualize your results” and “Overlay additional data” tab it is found the model of the first scheme 5.4. In this model, the tab panel is divided in a sidebar panel and a main panel. The Sidebar Panel is displayed with a greyish background colour and typically contains input controls. The Main Panel occupies 2/3 of the horizontal width and typically contains outputs. Other functions such as *fluidRow*(), *column()* and *box()* are used to build the layout up from a grid system (find a simplified version of the dashboard structure in B.4).

For the rest of tabs, “Run your query” tab and “Saved Results” tab, the tab panel is divided horizontally in two sections (see Figure 5.5). An upper fixed section that is a Fluid Row and contains input controls, and a Conditional Panel. The last element is displayed when the user has correctly followed the steps in the Fluid Row (see B.3).

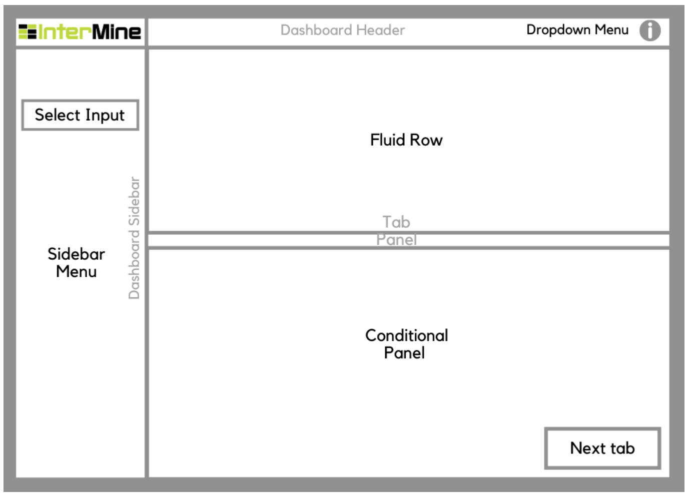


Figure 5.5: Second Layout configuration with a Conditional Panel.

Shiny offers many HTML *tag* functions for formatting text. The most used in this app are *p()* to create a paragraph, *br()* to create a line break, *hr()* to create a thematic break and *strong()* to give emphasis to the text. The tag *div()* has been used to placed action buttons, this is the case of the button on the button right corner to advance the tab (see in Figure 5.5 as Next tab).

From package `htmltools`, the functions *includeHTML()* and *includeMarkdown()* have been used to create the start screen. The first, *includeHTML()*, loads and renders the html file `intro_text.html`. This combined with the shiny functionality to display modal dialogs, makes a pop-up window with a welcoming message when the user enters to the app. The second function, *includeMarkdown()*, renders Markdown from the file `home.md` and turns it into HTML. This creates the "home" tab content, which includes an inline frame embedding a walk-through video of the whole app and some use-cases.

Once the architecture or user interface object has been described, the functionality defined in the server object will be reviewed going tab by tab.

5.2.3. Set Query Tab

This tab is in fact two sub-tabs, Templates and Query Builder. This means that the user can take two pathways, selecting a template query that can be modified or creating the query from scratch, see the details in appendix B.3.. The second one is only recommended for experienced users.

Regarding the Template tab, it is divided in three sections: the sidebar panel, the main panel, and the summary. A graphic description of the tab is schematized in Figure 5.6. The sidebar panel contains an informative message of the InterMine selected to start the query and a select list of all the template queries available for the InterMine (see in B.5). The list of templates is obtained from the function *getTemplates()* from the IntermineR package (see in B.6). In the main panel, the pre-defined constraints of the selected template are displayed (path, operator, and value) and the user can edit the values in a text area input control (see in B.7). Finally, in the summary section the *summary()* function from InterMineR returns a summary about the constraints in the form a data frame (see in B.10). Then, the *renderDataTable()* function from Shiny makes a reactive version of the data frame, which will be rendered with the DataTables library.

Regarding the Query Builder tab, when the user chooses it to start the query process a modal created by *shinyalert()* is displayed warning about the complexity of building a query from scratch. As in the Template tab an informative message of the InterMine selected is displayed. The tab is divided in two main sections, the query builder (see in B.8, and the summary. The first section can be seen in the following image (Figure 5.7).

The main functions are represented graphically in the diagram of Figure 5.8. Firstly, the data classes of the InterMine are displayed in the *select input control*, marked in blue with the number 1 in Figure 5.7. The options are retrieved from the "type" column in the resulting data frame *getModel()* (see B.6), a function from InterMineR. When the user selects a first data class, the attributes of it are displayed in a *select multiple input control* to define the data that wants to be shown, marked in green with the number 2. This also defines the type of sorting which will be used to order

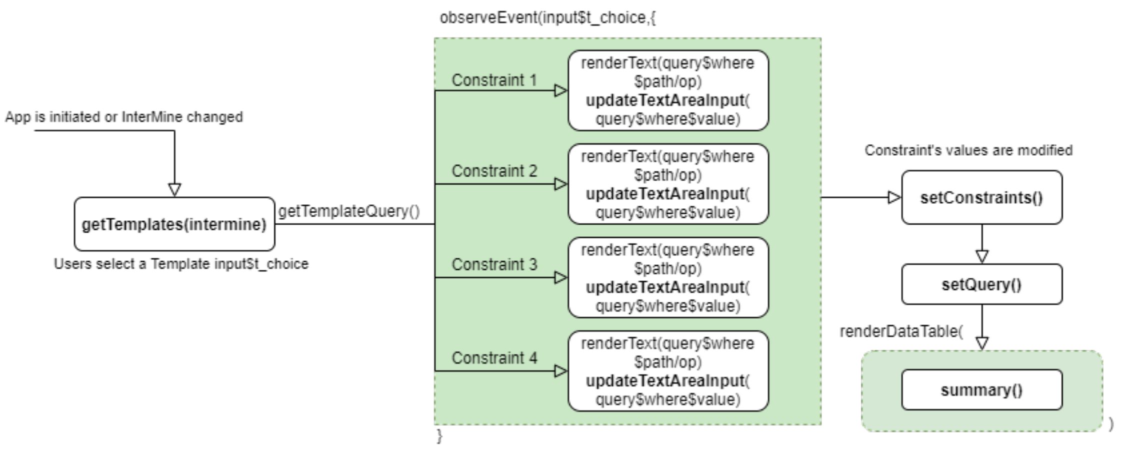

observeEvent(input$t_choice,{
App is initiated or InterMine changed
getTemplates(intermine)
getTemplateQuery()
Users select a Template input$t_choice
Constraint 1
Constraint 2
Constraint 3
Constraint 4
renderText(query$where $path/op) updateTextAreaInput(query$where$value)
Constraint's values are modified
setConstraints()
setQuery()
renderDataTable(
summary()
)
}

Figure 5.6: Diagram of the Template tab.

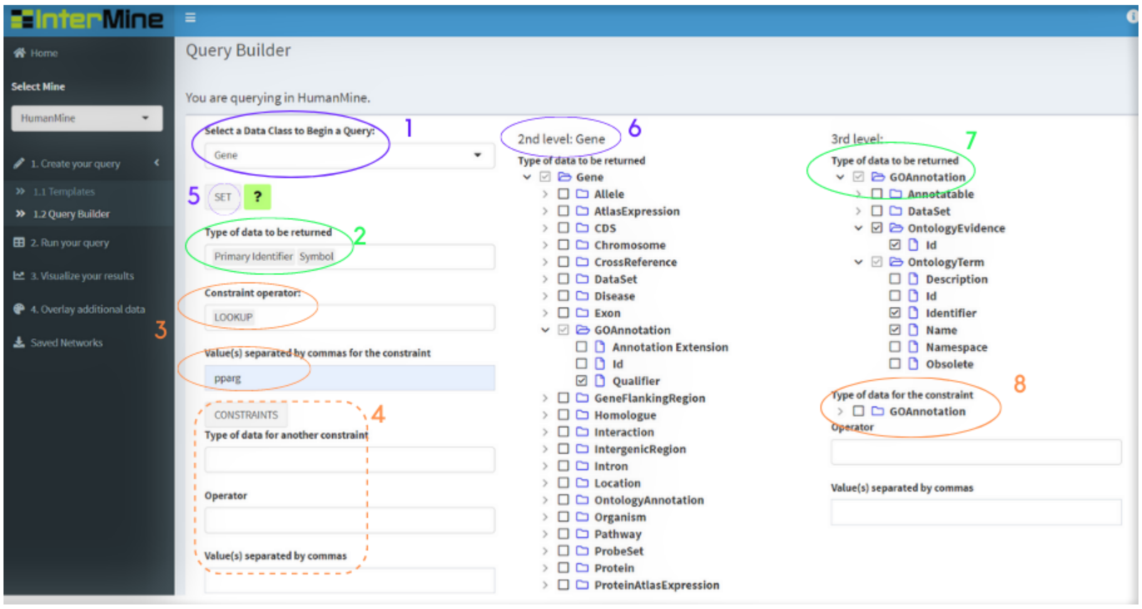


Figure 5.7: Screenshot of the Query Builder tab.

the retrieved data frame. The attributes are obtained from the `child_name` column from the *getModel* data frame. Only the `child_name` that correspond to a `type` equal to the data class just selected and with null `child_type` are retrieved (see Figure 5.8). At this point, the user has the first opportunity to set a constraint against the first data class. The constraint can be set in the input controls marked with the number 3. If the user presses constraints action button, a conditional panel that depends on this action appears below marked with number 4. In this new panel, the user can set a constraint against an attribute from the first data class.

To add a second level in the query, the user must press the button “SET” marked with number 5 in Figure 5.7. In the consecutive levels, one checkbox tree created with the `shinyCheckboxTree` package for the data type to be returned (shown in the results table), number 7, and another one for data types to set constraints, number 8. The user can overlay up to five levels by pressing the “SET” buttons that appear at the end of each checkbox tree. Once achieved the third level, the user can press "Overlay extra data" button to set two more levels into the query. The options that appear in each checkbox tree are obtained in the same manner; in the *getModel* resulting data frame are searched the rows with `type` equal to the previous level data class and that have a null `child_type` and from these rows the `child_type` values are retrieved. These values are transformed to paths applying some string operations and modifications. The whole process is done under *eventReactive()* and *observeEvent()* functions that respond to "event-like" reactive inputs and values.

Before discussing further aspects of this tab, reactive programming must be explained. In the Shiny framework the *observers* (functions) respond to any of their *reactive expression*, or inputs, changing. However, this is not always desirable. Sometimes it is wanted reactive values that trigger other calculations in this way, which are called *events* and are used under *observeEvent()* function. Instead, when what it is wanted is to create or update a value that only updates in response to an event, *eventReactive()* function is used. Using both functions, the events can be specified.

Finally, regarding the process of setting constraints, users can set multiple of them in each level. However, some rules that are specify in the documentation and in the same app must be followed such as separating the different values by commas. The constraints are created with the *setConstraints()* function from InterMineR (see B.8).

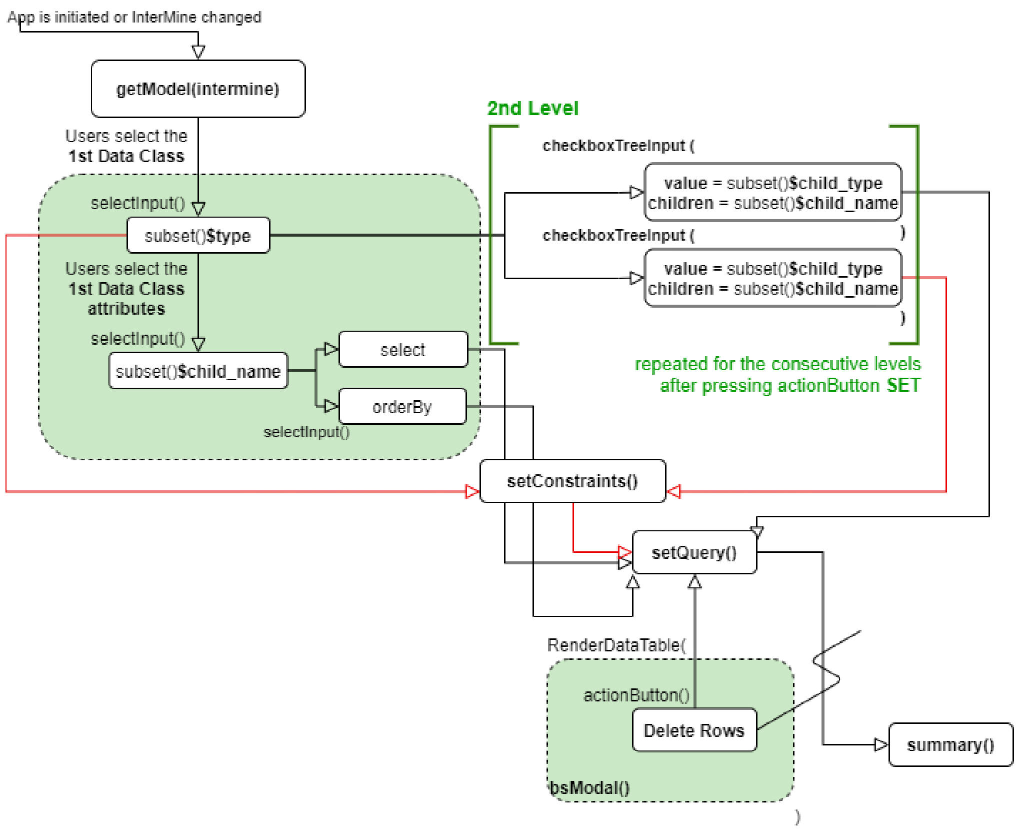

App is initiated or InterMine changed
getModel(intermine)
Users select the 1st Data Class
selectInput()
subset()$type
Users select the 1st Data Class attributes
selectInput()
subset()$child_name
select
orderBy
selectInput()
2nd Level
checkboxTreeInput (
value = subset()$child_type
children = subset()$child_name
)
checkboxTreeInput (
value = subset()$child_type
children = subset()$child_name
)
repeated for the consecutive levels after pressing actionButton SET
setConstraints()
setQuery()
RenderDataTable(
actionButton()
Delete Rows
bsModal()
)
summary()

Figure 5.8: Diagram of the Query Builder tab.

Before proceeding to the next tab, the user has the possibility to modify the paths (data types and attributes) that are going to be seen in the results table. When the user presses the "SET QUERY" button, a modal dialog is displayed with a data table where each row is a path (see in B.9). The user can delete the rows as is expressed in Figure 5.8. This has been included in response to the issue that when some data types of further levels are wanted other from first levels must be selected.

At the end of this process, and if the query has been correctly built using the function *setQuery()* from InterMineR, a summary of the constraints is shown (see B.10). Then, the *action button* to move to the next tab is enabled.

5.2.4. Run your Query Tab

Once the user has advance to this tab, the *runQuery()* function from InterMineR is called (see B.11). The resulting data frame is displayed, rendered by *renderDT()* and *datatable()* that create an HTML widget to display the data frame. It must be checked whether the data comes from the Template path or the Query Builder path. To do so, the *shinyalert()* function, the one creating the pop-up message in Query Builder tab, has an attribute called “callbackR”. This is an R function that will be called when the modal exits. In this case, the "callback" is to change a “reactive value” object, defined as *modality()*, to true. The value of "modality" is verified before using *datatable()* to display the results table. If the value is true it is known that the Query Builder path has been followed, else the Template path. And this is done from now on when the results of the query must be used.

To proceed, the user should select the “Set Nodes and Edges” button to select the id and the source for the Cytoscape Network Visualization. The options are the name columns from the results data table. At this point, the user may also want to set node’s attributes to be able to manipulate the network chart according to filtering criteria given by these attributes. In the following diagram 5.9 the flow described is represented.

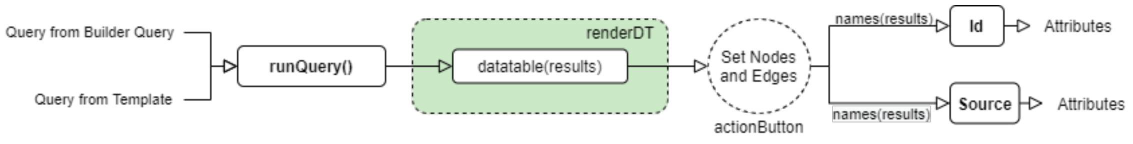


Figure 5.9: Diagram of the Query Builder tab.

5.2.5. Visualize your Results Tab

In this tab, the `cytoscape.js` JavaScript library is wrapped in a html widget for Shiny called *cyjShiny*. To prepare the data to be passed to the *cyjShiny()* function, which must be a graph in json format, another R package called `igraph` has been used. The function *graph_from_data_frame()* has been used to create an igraph graph from the data frame of results (see in B.12). Two data frames are passed to the function. It must be a data frame giving vertex metadata. The first data frame, nodes, is constructed from the id’s selected in the previous tab without repeating names. The first column of nodes (also known as vertices) data frame is assumed to contain symbolic vertex names, this will be added to the graphs as the "name" vertex attribute. The other data frame, edge data frame, is checked to contain only vertex names listed in vertices. This data frame consists of three columns. First column is source which corresponds to the nodes. Second column is target which corresponds to the edges. And third column is interaction defined as *source_target*. Using the functions *vertex_attr()* the node's attributes are included in the igraph and using *edges_attr()* the edge's attributes. The *igraph.to.graphNEL()* function is provided to convert the igraph to a graphNEL object. Finally, *graphToJSON()* is used to convert the graphNEL graph to a graph in json format (see in B.12). These steps are schematized in the left branch of diagram 5.10

Once the graph is prepared, it is time to examine the visualization options. Starting from the basic, what the user sees as default, the "cola" layout arranges the nodes using constraint-based optimization techniques (see in B.16). In the sidebar panel, the user can select a different layout, with *doLayout()*, and node(s) by ID or by attribute (see in B.13). With the selected nodes the user can decide to do different actions using the buttons in the main panel. These functionalities are provided by the R package cyjShiny. The visualization options are zoom the selected nodes with *fitSelected()*, reset the view with *fit()*, select the first neighbour(s) with *sfn()*, invert the selection with *invertSelection()*, unselect nodes with *clearSelection()*, remove selected nodes *hideSelection()*, show all again with *showAll()* and get the selected nodes names with *getSelectedNodes()* (see in B.14). The nodes that are removed are kept in this condition in the next tab taking advantage of creating a reactive value that changes dynamically with *reactiveVal()* (see in B.15).

Finally, the user can save the network as an image in PNG format. This functionality is provided by the function *savePNGtoFile()* by cyjShiny package. Using the function *shinyalert()* a modal dialog informs about the download directory location (see in B.16).

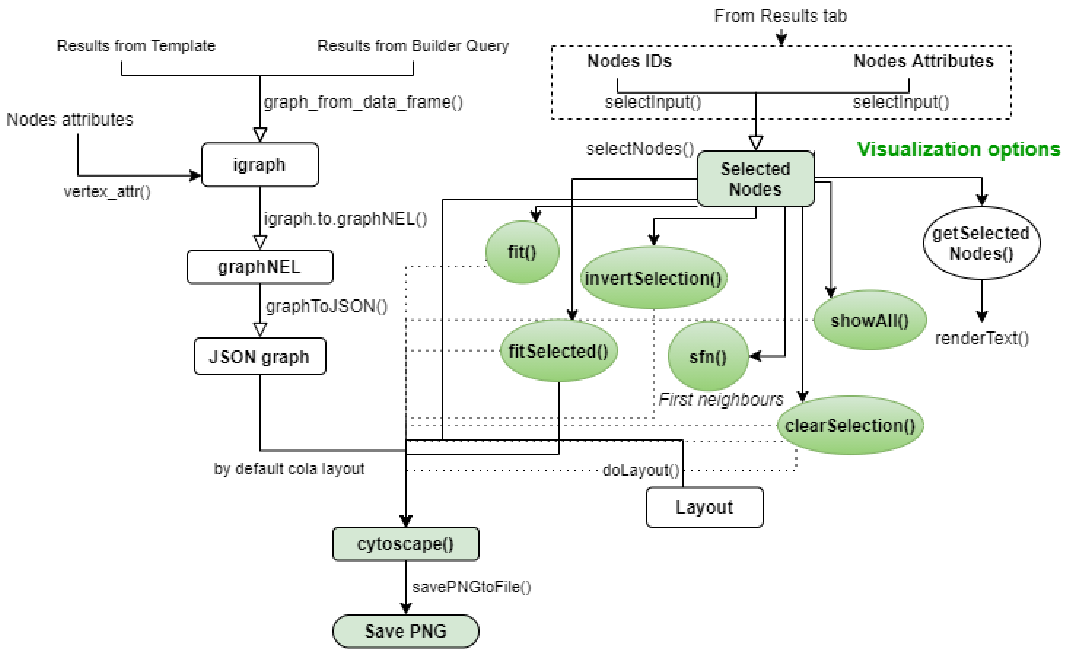


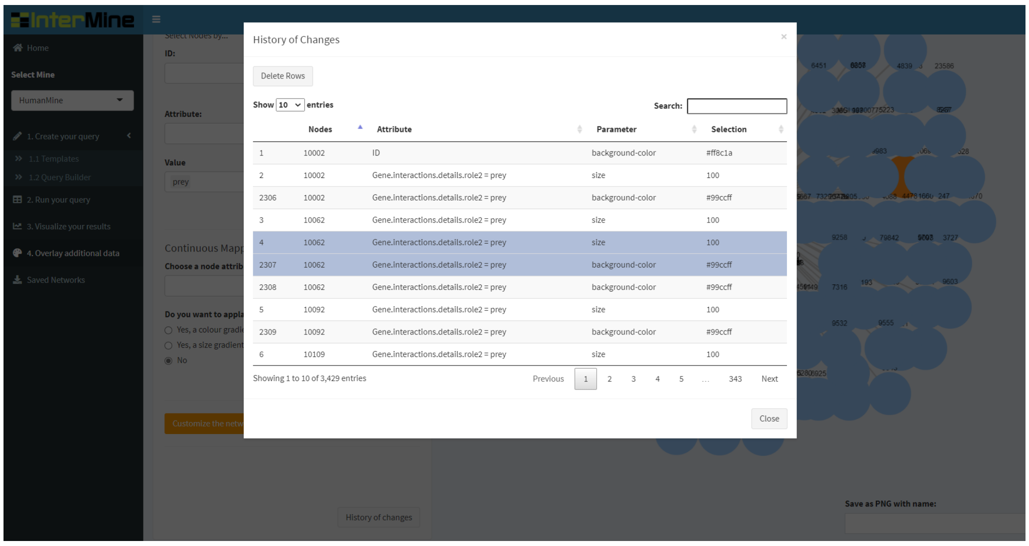
Figure 5.10: Diagram of the Visualize your Results tab.

5.2.6. Overlay Additional Data Tab

In this tab the user has different options at its disposal to style the network chart. The network chart is created using the function *cytoscape()* from the R package cytoscape, a html widget for cytoscape.js (see in B.20). Two data frames, one for nodes and the other for edges, must be passed to the function (see in B.17). The nodes data frame has as components *id*, *node_color*, *node_width*, *node_height* and *node_shape* (see in B.19). The last four components are parameters to style the node's body using the function *node_style()* (see in B.21). The edges data frame components are *source* (corresponds to nodes), *target* (corresponds to edges) and *interaction*. The configuration of the two data frames can be seen in the left side of the diagram from Figure 5.12. The layouts are specified using *cola_layout()* and *layout()*. The function *panzoom()* enables for panning (panoramic movement) and zooming in and out (see in B.20).

In the sidebar panel the user can either customize the node's body or apply a gradient. Customize the node's body means that selecting the nodes by ID or by attribute value the user can change their shape, size, or colour (see the options on the right side of Figure 5.12). If instead the user decides to apply a gradient, defines a continuous-to-continuous mapping and continuous data are mapped to properties. Depending on the property it can be distinguished between mapping continuous numerical values to a colour gradient or to node size (see in B.18).

Each time the user defines a new style the "Customize the Network" button must be pressed to apply the changes (see in B.21). The styles are saved and can be seen pressing the "History of changes" button. When this is done, a modal window appears with a table of four columns: nodes, attribute, parameter, and selection (see Figure 5.11). Each row is the style of one node for a concrete parameter that can be background-colour, size, or shape. The user can select one or multiple rows to delete, this way returning to the default values of the parameters.


History of Changes

Delete Rows

Show 10 entries Search:

	Nodes	Attribute	Parameter	Selection
1	10002	ID	background-color	#ff8c1a
2	10002	Gene.interactions.details.role2 = prey	size	100
2306	10002	Gene.interactions.details.role2 = prey	background-color	#99ccff
3	10062	Gene.interactions.details.role2 = prey	size	100
4	10062	Gene.interactions.details.role2 = prey	size	100
2307	10062	Gene.interactions.details.role2 = prey	background-color	#99ccff
2308	10062	Gene.interactions.details.role2 = prey	background-color	#99ccff
5	10092	Gene.interactions.details.role2 = prey	size	100
2309	10092	Gene.interactions.details.role2 = prey	background-color	#99ccff
6	10109	Gene.interactions.details.role2 = prey	size	100

Showing 1 to 10 of 3,429 entries

Figure 5.11: History of Changes modal dialog where changes can be seen and deleted.

Regarding the definition of gradients, the user must select a node attribute from the ones chosen in the Run your Query tab. Then, the user can specify the range of values within the values of the attribute to map and choose between a size or a colour gradient (see in B.18). The size gradient can be set between 10 and 200 in pixels at zoom 1. And the colour gradient must be defined choosing two extreme colours from a basic pallet, using *colorSelectorInput()* function from ShinyWidgets package (see in Figure 5.12.

Finally, the user can save the results as an image in PNG format or in a ZIP folder with the files necessary to display again an interactive Network chart. For the first, the function *webshot()* from the same name package is used along with the Shiny *downloadHandler()* (see in B.22). The ZIP folder will contain the results of the query in a CSV file, two CSV files with the basic components and the modifications made to the Network and a JSON file of the Network. These files are created using *write.csv()* for the CSV files. For the JSON file the function *dataFramestoJSON()* from RCyjs package is used to convert the nodes and edges data frames into a JSON format network. Then, the files are zipped using *zipr()* from zip package (see in B.22). Just to be mentioned, if no names is provided by the user for the downloading of the ZIP folder, a default names is generated beginning with *workflow_final* and following with the date.

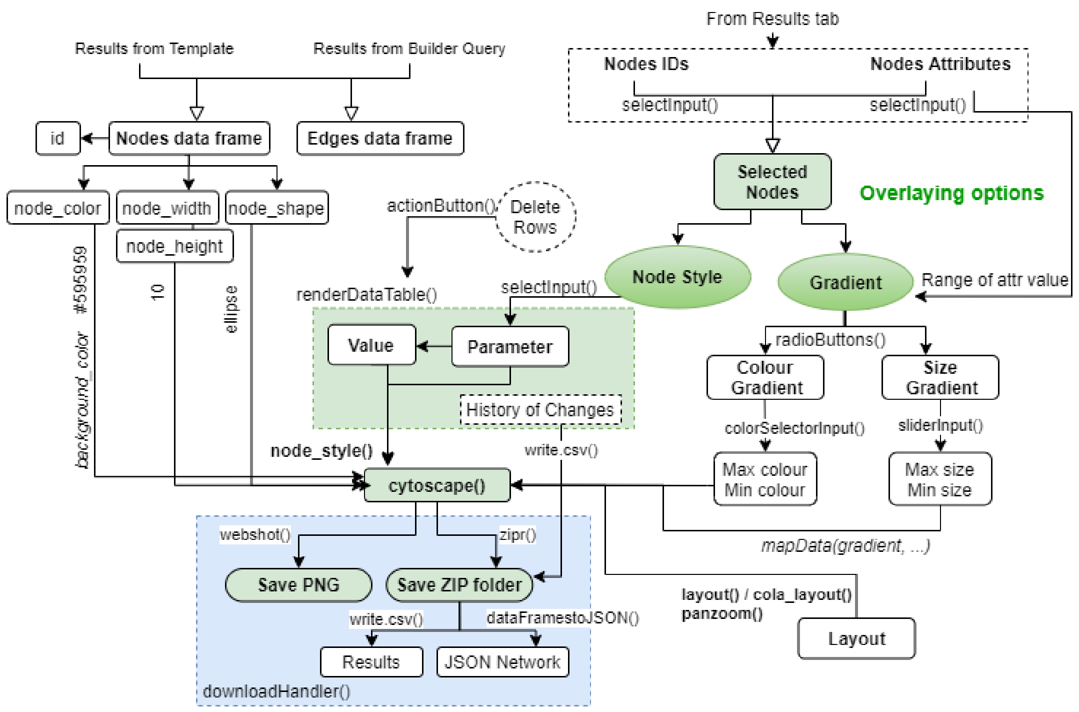


Figure 5.12: Diagram of the Overlay Additional Data tab.

5.2.7. Saved Networks tab

In this last tab, the user can display previously saved networks. Using *fileInput()* function from shiny, a file upload control is created. Pressing "Browse" the user can navigate to find the ZIP folder that have been saved in the "Overlay additional data" tab. Once it is uploaded, the user must press Unzip files action button, and this will call the function *unzip()* that extract the files from the folder. Using packages `DT` and `data.table` the results from the query are displayed in a table as have been seen in Run your Query tab (see in B.23). Using the package `cytoscape` the interactive Network Chart is displayed. The visual styles for the node's body set by the user are maintained and appear in the visualization. The layout by default is cola. Figure 5.13 shows a diagram with the workflow of the tab.

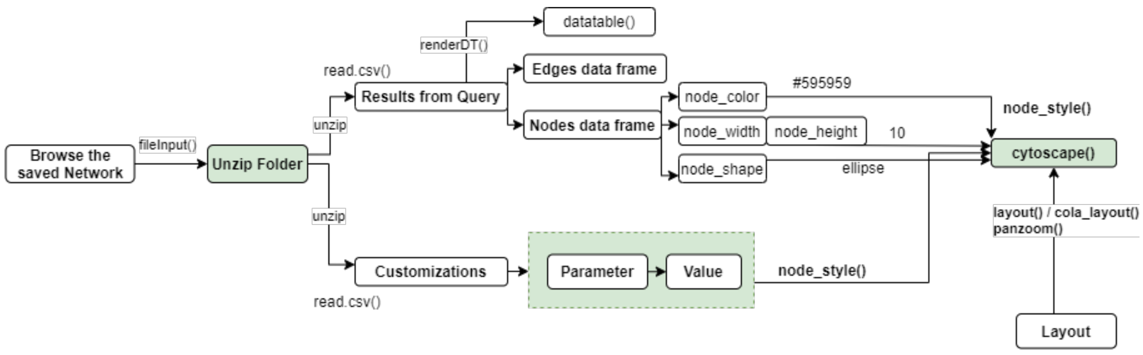


Figure 5.13: Diagram of the Saved Networks tab.

5.2.8. Customer Support

Having a great support is as important as having a great product. When software accomplishes both product and support, the customer has a great experience. In this app three support channels have been considered: developer contact and feedback, help inside the app and documentation.

As mentioned before in this section, the dropdown menu, present in all the tabs, gives access to the GitHub repository where the source code is published and to the Issues section of the repository. The last element, Issues, is a bug tracker where the user can be in contact with the developer and the messages are shared with the other users. Thus, Issues is a great way to collect user feedback and report software bugs.

Regarding the app help system, help content should be easy for users to find and give answer to common questions about the app. For these reasons, it has been decided to include a help button in each tab and hints in the different elements. The help buttons can be easily recognized by a question mark icon. When the user presses the help buttons step-by-step introductions appeared. The `rintrojs` R package, based on the JavaScript library `Intro.js`, is used to add these instructions. The function *introjsUI()* must be called once in the *ui* and then the function *rintrojs()* supports programmatic introductions and dynamically generate them using the steps option. Tip or hints are given using the package `shinyBS`. With the function *bsTooltip()* inside the UI, information is added to controls and outputs. Hovering with the mouse cursor over these controls and outputs the information is displayed. Additionally, some feedback messages are only displayed in reaction to an action. Those are the feedback dangers created with the function *showFeedbackDanger()* from the package `shinyFeedback`. When the condition to display the message is not fulfilled any more the function *hideFeedback()* is called to hide it.

Finally, documentation takes the form of a user guide. It has been written in Latex. The contents of it include the requirements, the capabilities, basic usage instructions and where to find the source code. The requirements are mainly software, no hardware requirements are necessary except for a computer with storage to save RStudio. The user guide can be read in the appendix B.5..

5.2.9. GitHub Repository

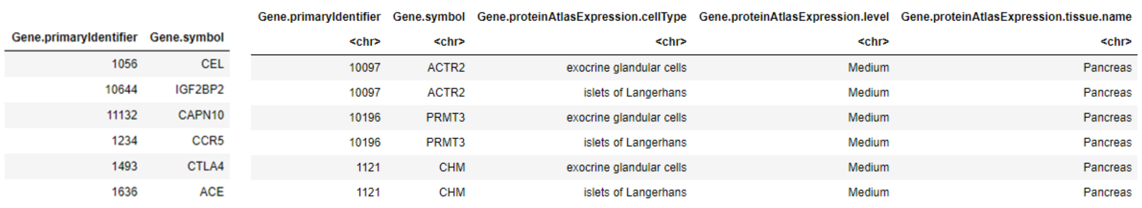
Source code is open and is held in the git repository in GitHub. The users can download the last version from GitHub following the indication provided in the README.md.

5.3. Results

5.3.1. Improvements of the core of InterMineR

To evaluate the proper functioning of the new version of the InterMineR package a practical case has been followed. This consists of building and running two queries and intersect their results. Firstly, *initInterMine()*, which import the Service class, is used to say that HumanMine is the mine wanted for querying and set the API Access key or Token. The first query selects all the genes associated with diabetes. This requires two constraints, first ensure that all genes returned are Homo Sapiens genes (HumanMine contains some non-human genes for homology query purposes) and second restrict results to genes that are associated with diabetes. The second query is for genes that are in the public HumanMine list PL_Pax6_Targets, that are also expressed in the pancreas at a High or Medium level, according to data originally from the Human Protein Atlas Project [27]. In this case, the constraints are three. First, all the genes should be in the list PL_Pax6_Targets, that is the same that saying that all the genes must be targets of Pax-6. Second, Gene.proteinAtlasExpression.tissue.name should be equal to Pancreas. And third, Gene.proteinAtlasExpression.level should be set to High OR Medium. This will require two constraints, one for each of medium and high. You can find the code in appendix A.4.

These are the results from the queries:

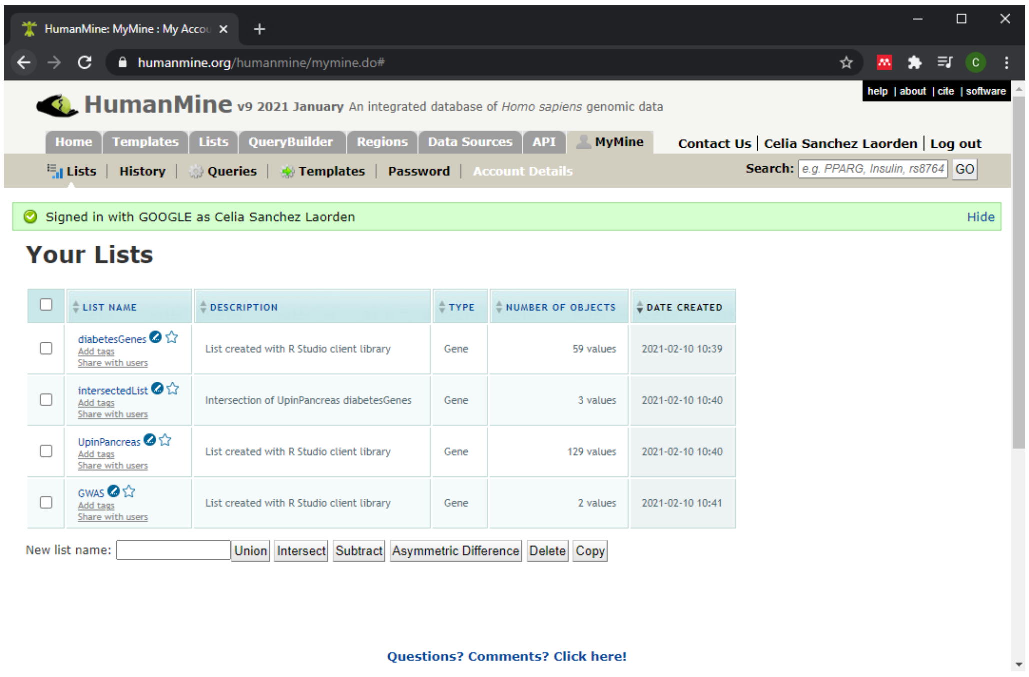


Gene.primaryIdentifier	Gene.symbol
1056	CEL
10644	IGF2BP2
11132	CAPN10
1234	CCR5
1493	CTLA4
1636	ACE

Gene.primaryIdentifier	Gene.symbol	Gene.proteinAtlasExpression.cellType	Gene.proteinAtlasExpression.level	Gene.proteinAtlasExpression.tissue.name
<chr>	<chr>	<chr>	<chr>	<chr>
10097	ACTR2	exocrine glandular cells	Medium	Pancreas
10097	ACTR2	islets of Langerhans	Medium	Pancreas
10196	PRMT3	exocrine glandular cells	Medium	Pancreas
10196	PRMT3	islets of Langerhans	Medium	Pancreas
1121	CHM	exocrine glandular cells	Medium	Pancreas
1121	CHM	islets of Langerhans	Medium	Pancreas

Figure 5.14: Results of queries: diabetes genes (left) and Pax6 targets that have high expression in the Pancreas (right).

From both results, the primary identifiers are saved in lists in the account. The names given to these lists are diabetesGenes and UpinPancreas. At this point, it has been tested *list_manager()*, *delete_lists()* and *create_list()*, and indirectly *GET_api_list()*.


HumanMine: MyMine : My Accou
humanmine.org/humanmine/mymine.do#
help | about | cite | software
HumanMine v9 2021 January An integrated database of *Homo sapiens* genomic data
Home
Templates
Lists
QueryBuilder
Regions
Data Sources
API
MyMine
Contact Us | Celia Sanchez Laorden | Log out
Lists | History | Queries | Templates | Password | Account Details
Search: e.g. PPARG, Insulin, rs8764 GO

Signed in with GOOGLE as Celia Sanchez Laorden Hide

Your Lists

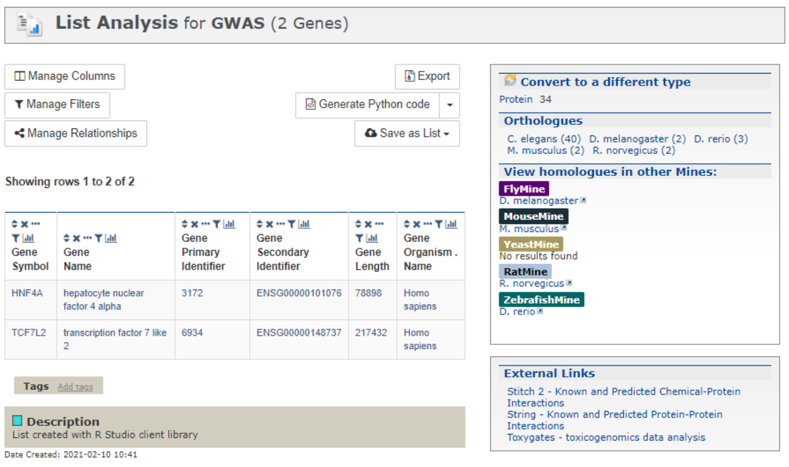
	LIST NAME	DESCRIPTION	TYPE	NUMBER OF OBJECTS	DATE CREATED
☐	diabetesGenes Add tags Share with users	List created with R Studio client library	Gene	59 values	2021-02-10 10:39
☐	intersectedList Add tags Share with users	Intersection of UpinPancreas diabetesGenes	Gene	3 values	2021-02-10 10:40
☐	UpinPancreas Add tags Share with users	List created with R Studio client library	Gene	129 values	2021-02-10 10:40
☐	GWAS Add tags Share with users	List created with R Studio client library	Gene	2 values	2021-02-10 10:41

Figure 5.15: Your Lists tab in MyMine space from HumanMine.

Next, the intersect method is used to find those genes that are upregulated in the pancreas that are also associated with the disease diabetes. The first (UpinPancreas) and second (diabetesGenes) lists are intersected using the method from the `ListManager` class. The results are genes HNF 1B, HNF 4A and TCF7L2 as can be seen in the resulting list created in MyMine section or the personal account, on the website. Until here, it has been tested indirectly the function *do_operation()*.

Finally, we fed the intersected list from above back into another query to see if there was any association of these genes with diabetes phenotypes according to GWAS (Genome-wide association studies) from National Human Genome Research Institute (NHGRI). GWAS studies seek to associate single nucleotide polymorphisms (SNPs) with specific phenotypes and diseases and have uncovered scores of genetic variants associated with complex disease traits. To do so, the three primary identifiers resulting of the intersection operation are the new constraints plus the diabetes phenotype condition. The unique genes that are returned are saved in a new list, called GWAS, in the personal account. These genes are HNF 4A and TCF7L2.

The Jupyter Notebook for R with all workflow for this practical case can be found in the appendix A.4..


List Analysis for GWAS (2 Genes)

Showing rows 1 to 2 of 2

Gene Symbol	Gene Name	Gene Primary Identifier	Gene Secondary Identifier	Gene Length	Gene Organism . Name
HNF4A	hepatocyte nuclear factor 4 alpha	3172	ENSG00000101076	78898	Homo sapiens
TCF7L2	transcription factor 7 like 2	6934	ENSG00000148737	217432	Homo sapiens

Description
List created with R Studio client library
Date Created: 2021-02-10 10:41

Convert to a different type
Protein 34

Orthologues
C. elegans (40) D. melanogaster (2) D. rerio (3)
M. musculus (2) R. norvegicus (2)

View homologues in other Mines:
FlyMine
D. melanogaster
MouseMine
M. musculus
YeastMine
No results found
RatMine
R. norvegicus
ZebrafishMine
D. rerio

External Links
Stitch 2 - Known and Predicted Chemical-Protein Interactions
String - Known and Predicted Protein-Protein Interactions
Toxygates - toxicogenomics data analysis

Figure 5.16: List Analysis for GWAS page showing the two genes resulting from the intersection.

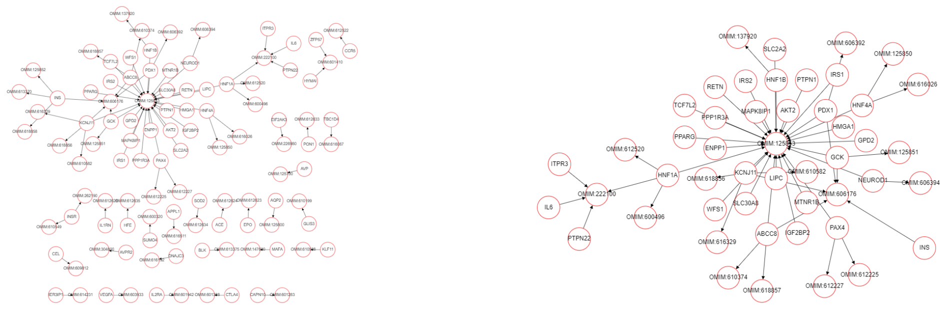
5.3.2. Use cases for the InterMineR-Cytoscape Shiny Interface

In this section the applications are explored through two use-cases. For this, two InterMines have been examined; HumanMine and CovidMine.

HumanMine integrates many types of data for Homo sapiens, but genomic is the most representative type. For the demonstration, Disease to Gene mappings from OMIM (Online Mendelian Inheritance in Man) have been acquired following the template "Disease –> Genes + RNA-seq Expression". This template shows the genes involved and their tissue-specific expression level for a specified disease, in this case Diabetes. Using the Query Builder, it has been added more restrictive constraints to the *Disease.name* and the *Gene.symbol*.

Besides, CovidMine is a test InterMine dedicated to SARS-CoV-2 genomics and geographic distribution data. It integrates confirmed COVID-19 cases, deaths, new confirmed cases and new deaths for countries from Our World In Data, SARS-CoV-2 reference genome and nucleotide sequences from isolates deposited in Genbank. For the exhibition, the template "Date –> Confirmed cases, deaths, new confirmed cases and new deaths" is going to be used.

For the HumanMine use-case, the researcher investigating genes involved in different types of Diabetes should go to the Template Query tab and select the template "Disease –> Genes + RNA-seq Expression" (see in appendix C.1). By default, this template constraints the results for Diabetes disease as seen in appendix C.2. The researcher would see the constraints in the summary table, see in appendix C.3. Moving to the next tab (Query Results), the nodes, edges and attributes should be determined. The nodes and edges are determined choosing the source and target data. The source nodes interact with the target nodes, drawing the edges. To see a map with all the types of Diabetes and the genes involved, the researcher should select as source diseases' primary identifiers and as target genes symbols (see in appendix C.4). In the visualize your results tab (C.5), the researcher wants to focus on the genes that are related with Type 2 Diabetes Mellitus and see the other diabetes that share genes with it. To do so, the easiest path is to click on the node *OMIM:125853* (Type 2 Diabetes Mellitus) and click on "Select First Neighbor" button as many times as interconnection levels the researcher desires. It can be seen in C.7 and C.8. Then, by clicking on "Invert Selected" and "Remove Selected" buttons consecutively, a new filtered network is obtained (see in C.9 and C.10). The new network shows the genes involved with Type 2 Diabetes and the other types of Diabetes where they are also involved. It can be downloaded as an image in PNG format, as can be seen in appendix C.12 and C.13. The following are the original and the new networks:



(a) Original. (b) Filtered by First Neighbours.

Figure 5.17: Networks of the Workflow A (see them bigger in C.14).

Moving to the Overlay additional data, the researcher can apply more filters to make the resulting network from the last tab more understandable. The modifications that have been done are: changing the background colour of the genes involved in Type 2 Diabetes Mellitus to orange and changing the shape of the genes associated with Type 1 Diabetes Mellitus to triangular. The steps can be followed in the appendixes C.16, C.17 and C.18. The resulting network can be seen in Figure 5.18.

Finally, it is recommended to save the network in a ZIP folder as can bee seen in appendix C.19. This way, the researcher could uploaded from the last tab of the interface, the Saved Networks tab, and visualize the results again (see in C.20).

With the workflow A, it can clearly be see that only one gene, HNF1A, appears to be involved in both diseases. The researcher has seen the need to add more restrictive constraints. In the next workflow, the Query Builder is used with that purpose. The researcher wants to know the level of expression in the different tissues of the genes related with Type 1 and Type 2 Diabetes Mellitus.

Firstly, the researcher should specify a data class to begin the new query, in this case *Disease*. Then, *Disease* type of data to be returned is selected, in this case *Name* and *Primary Identifier*. In this first data class a constraint to *Disease.name* is set indicating that only results of Type 1 Diabetes Mellitus are wanted. How is done the configuration of the first data class can be seen in the appendix C.21. The configuration for the following levels can be visualized in the video for this use-case.

When the researcher runs the query, a table with 216 entries is displayed. This table contains information about the four genes involved in Type 1 Diabetes Mellitus and its expression level in different tissues. As the researcher wants a general view of the expression of all four genes, the expression score column should be selected as target data and the gene symbol column as source (see in appendix C.22). The

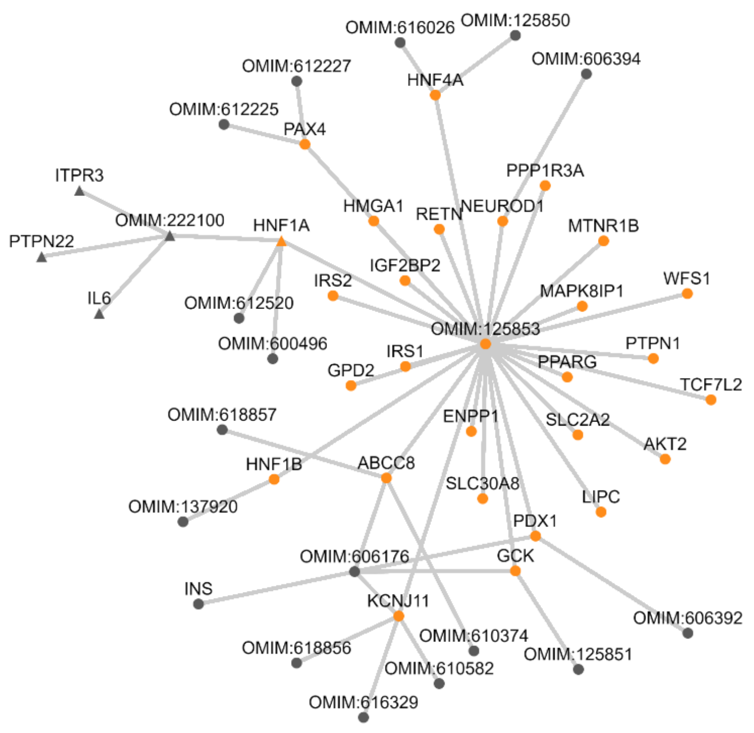


Figure 5.18: Customized network of Workflow A.

researcher can plot a quick size gradient map going to the Overlay additional data tab. The results are shown in the appendix C.24.

It is at this point that the researcher goes back to the Query Builder tab to add another constraint for *Gene.symbol* as can be seen in the appendix C.25. The reason to do that is restricting the results to one of the four genes to obtain a map of the expression level in the different tissues for each gene. And once, the four maps are being created compared them.

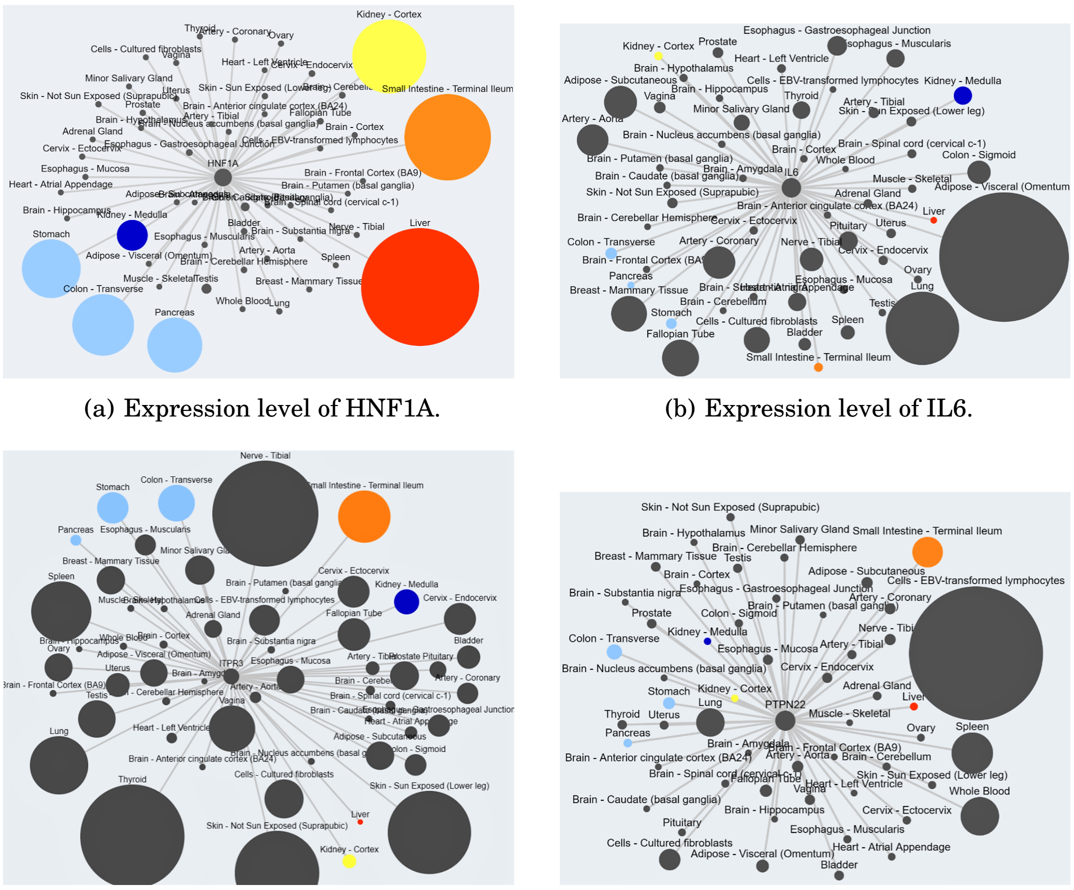
The summary of constraints for the query of HNF1A gene expression in Diabetes Mellitus Type 1 can be seen in the appendix C.26. Once the researcher runs the new query, the results are displayed on a table of 54 entries, one for each different tissue. At this point, the target and source data is defined by *RNASeqResults.tissue* column and *Genes.symbol* column, respectively. As nodes attributes, it is necessary to have *RNASeqResults.expressionScore*. This configuration can be seen in the appendix C.27.

Again, a size gradient map is plotted. After that, five background-colour overlays are set for the tissues with highest level of expression of HNF1A gene (as seen in C.28). The summary of overlays is displayed pressing the "History of Changes" button, and this one can be found in the appendix C.29.

The expression data is not disease-specific as can be seen in the appendix C.30, where the HNF1A gene expression level is mapped for Type 1 and Type 2 Diabetes Mellitus. It must be mentioned that as the overlaying changes are saved, unless the user wants to delete them in the History of Changes window, the customized background colour configuration is preserved.

In the appendix C.31 - C.36, the constraints summaries for other three queries (genes

IL6, *ITPR3*, *PTPN22*) and the size gradient mapping configuration are shown.



(a) Expression level of HNF1A.

(b) Expression level of IL6.

(c) Expression level of ITPR3.

(d) Expression level of PTPN22.

Figure 5.19: Size gradients for the genes involved in Type 1 Diabetes Mellitus.

From Figure 5.19(a), it can be seen that the HNF1A is mostly expressed in the liver (in red), followed by the small intestine - terminal lleum (in orange) and the kidney cortex (in yellow). Then, at the same level, there are the stomach, the transverse colon and the pancreas (in blue), and finally the kidney medulla (in dark blue). The expression score of the HNF1A in the liver is rounded 8.7 (see the maximum value in the range of values to map of appendix C.30(a)). Comparing this value with the highest ones of the other three genes, it is concluded that the HNF1A is the less expressed of the four genes. However, HNF1A plays an important role in the maturity onset diabetes of the young (MODY) type 3 [28]. MODY type 3 has the primary identifier *OMIM:600496*. Looking at Figure 5.18, it is the only gene involved in this type of diabetes.

Moving to IL6 gene map 5.19(b), it is very highly expressed in adipose tissue (Adipose-Visceral). IL6 plays a role in the inflammation of adipose tissue in Type 1 Diabetes Mellitus [29]. Looking at the preliminary conclusions of a study that investigates the relation between be affected by Diabetes Mellitus and the increment of severity of coronavirus disease[30], it points us to the high level of expression of IL6 in the lungs that can be seen in the network 5.19(b).

The ITPR3 gene has the highest expression score, being this one 135.751 in the nerve-tibial but closely followed by thyroid with 133.338, and skin with 107.076

for not sun exposed and 105.507 for sun exposed. The expression data presented here does not provide any evidence for its involvement in key tissues. However, its role has been related with mitochondrial dysfunction and ROS production that accelerates diabetes [31].

Finally, in the map for PTPN22 gene 5.19(d), the EBV (Epstein-Barr virus) transformed lymphocytes have the highest score with a value of 50.562. Although they are not the highest factor of risk, viruses such as Epstein Barr or Coxsackie can be the cause of Type 1 Diabetes [32].

So far, it has been reviewed a relevant case for biomedicine research application of the interface. Now, moving to an easiest use-case but very relevant in the pandemic context, we will take a closer look to the situation in each country.

From the sidebar menu, the CovidMine is selected. Then, in the Template Query tab the "Date –> Confirmed cases, deaths, new confirmed cases and new deaths" template. It is going to be chosen a nearest date to reduce the number of results returned. So, it is selected a week going from 13th April to 19th April 2021 (see in the appendix C.37). In the next tab, the results are displayed on a table with 1,467 entries. To create a network with the countries that will be mapped by confirmed cases and deaths, *GeoLocation.country* columnn is the target data and *GeoLocation.cases.totalConfirmed*, *GeoLocation.cases.totalDeaths*, *GeoLocation.cases.newConfirmed* and *GeoLocation.cases.newDeaths* are the nodes attributes that are going to be used to apply a gradient (see in C.38).

In the Visualize your Results tab, the node *2021-04-14* and its first neighbours are selected using the buttons (C.39). Then, the selection is inverted (C.40) and finally the new selection is removed (C.41). The latest selection is preserved in the next tab. A size gradient is applied to the node attribute *GeoLocation.cases.newConfirmed* but the presence of continents in the network distorts the result (see in the appendix C.42). At this point, going back to the Visualize your Results tab the continents are selected and removed (C.43-C.44). The new size gradient without continents nodes can be found in the appendix C.45. It can be fast and clear that India was the country with bigger number of new confirmed cases in the middle of April 2021. India was followed for the United States of America, Brazil and Turkey.

Another type of gradients are the colour gradient. Selecting the range of values to map in the options for continuous mapping it can be specified a threshold to a numeric attribute. For example, for the case of new deaths per day the threshold has been set to 500 (see in C.46(b)). The resulting network shows in red the countries where the threshold was exceeded. Again, these countries were India, the United States and Brazil. Also, in red, it could be found Poland, Italy, Mexico and Ukraine. With a darker red or purple were Turkey, Russia, Argentina, Peru, Hungary, Colombia, Iran, Germany and France. The rest of countries were clearly determined to be under 500 new deaths.

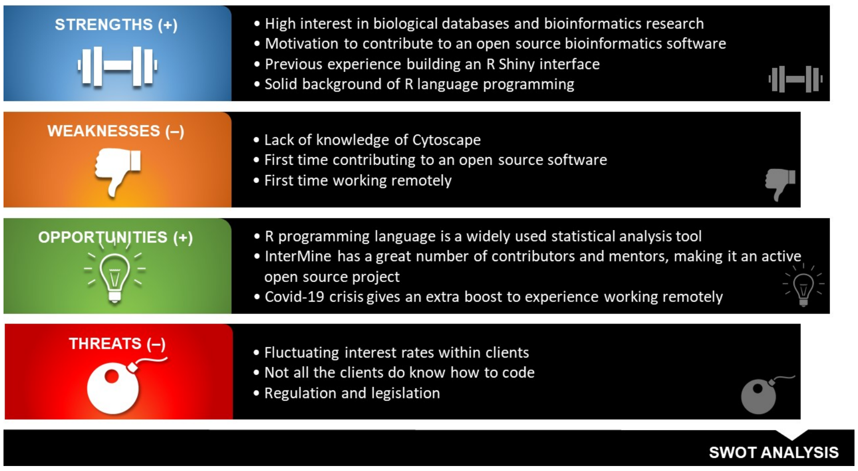
Finally, only the continents have been selected to show four basic networks C.47 where the new and total confirmed cases and deaths on 14th April 2021 are represented. The network for the new confirmed cases has been customized with different background colours following a colour scale where the warm colours are for the countries with highest number of cases and the cold colours for the opposite extreme (see in C.48). This overlay network has been saved in a ZIP folder. Going to the last tab of the Shiny interface, the previously ZIP folder has been uploaded, unzipped the files, and the network with the customization of colours has been displayed (see in C.49).

These use-cases can also be followed in the introductory videos from the Home tab of the Shiny interface or the YouTube channel.

CHAPTER 6. ORGANIZATION

6.1. Technique pre-feasibility study

It is important to detect the strengths, weakness, opportunities, and threats of the project and go a step further to create actionable strategies and plans to improve. The background in R language and R Shiny interfaces brings a competitive advantage to this student project but also to the InterMine improvement. The context of an open source software project also offers a great opportunity for cooperation and helps building strategies to minimize the weaknesses and threats. The following figures are the SWOT analysis matrix in 6.1 and a TOWS analysis in 6.2.



STRENGTHS (+)
- High interest in biological databases and bioinformatics research
- Motivation to contribute to an open source bioinformatics software
- Previous experience building an R Shiny interface
- Solid background of R language programming

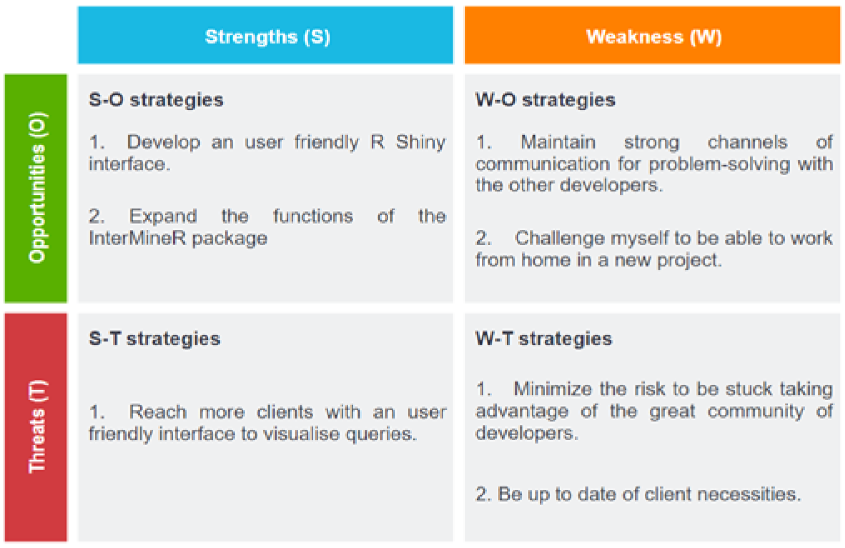
WEAKNESSES (–)
- Lack of knowledge of Cytoscape
- First time contributing to an open source software
- First time working remotely

OPPORTUNITIES (+)
- R programming language is a widely used statistical analysis tool
- InterMine has a great number of contributors and mentors, making it an active open source project
- Covid-19 crisis gives an extra boost to experience working remotely

THREATS (–)
- Fluctuating interest rates within clients
- Not all the clients do know how to code
- Regulation and legislation

SWOT ANALYSIS

Figure 6.1: SWOT Analysis.



	Strengths (S)	Weakness (W)
Opportunities (O)	**S-O strategies** 1. Develop an user friendly R Shiny interface. 2. Expand the functions of the InterMineR package	**W-O strategies** 1. Maintain strong channels of communication for problem-solving with the other developers. 2. Challenge myself to be able to work from home in a new project.
Threats (T)	**S-T strategies** 1. Reach more clients with an user friendly interface to visualise queries.	**W-T strategies** 1. Minimize the risk to be stuck taking advantage of the great community of developers. 2. Be up to date of client necessities.

Figure 6.2: TOWS Analysis.

6.2. Schedule of Execution

From the timetable 6.3 an overview of the scheduled tasks during the development of the project, indicating the deadlines for each of them and their duration in days, is seen. In the appendix D.2, the Work Breakdown Structure (WBS) can be read. In the WBS the project is splitted in smaller components to be the guidance of the schedule development and control. The start date of the execution of the tasks was June 22, and the end of the internship it was scheduled for August 21. It was planned to take a break between July 10 and July 20. Also, it was planned to continue polishing details during the first semester.

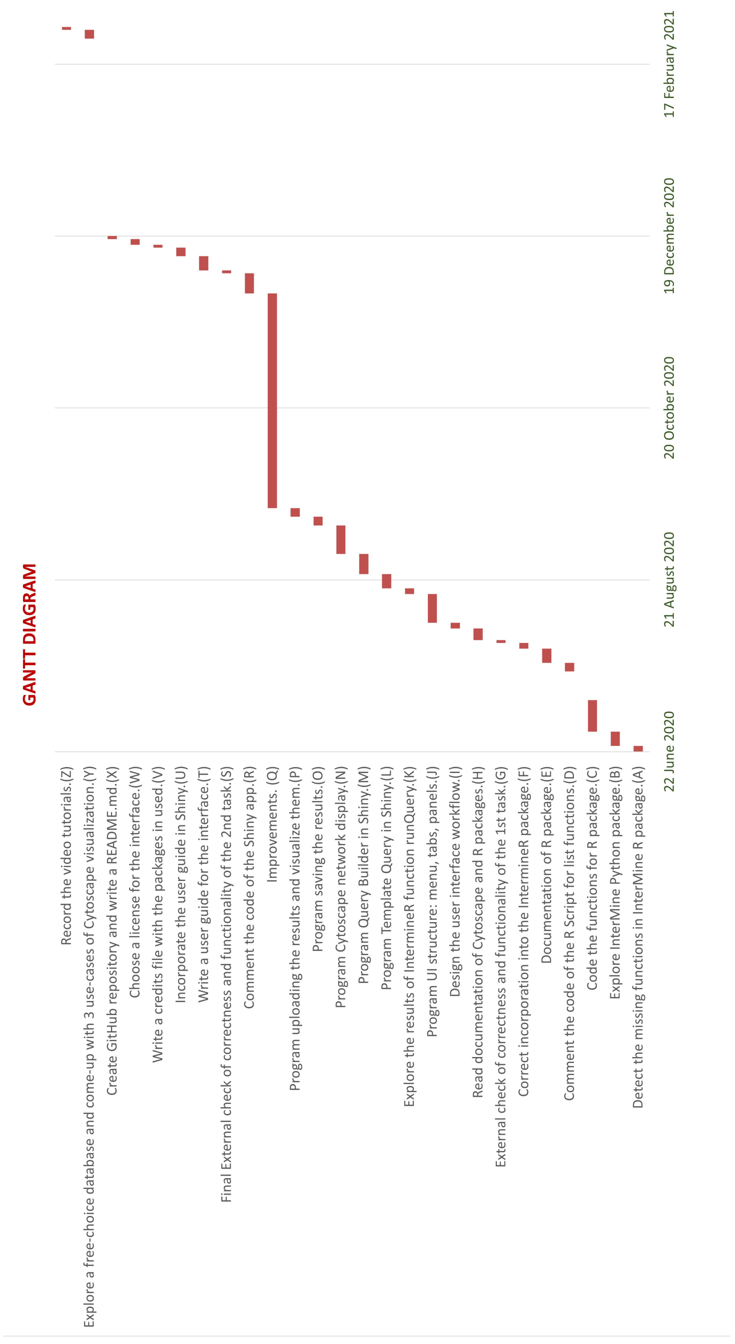


Figure 6.3: GANTT Chart.

CHAPTER 7. ECONOMIC PRE-FEASIBILITY STUDY

7.1. Cost study

In this section is considered the cost derived from the work of the software developer, or the intern student, and the supervisors. The corresponding labour costs are described in the following table:

LABOUR	WORKING HOURS	UNITARY COST (£/h)	NATIONAL INSURANCE (£)	TOTAL COST (£)
DEVELOPER	442	16.82	892.13	8,326.57
SUPERVISOR	27	31.26	101.28	945.30
SUBTOTAL				9,271.87

Table 7.1: Labour costs.

In the table 7.1, it has been considered the salary of a Biomedical Engineer and of a Senior Researcher in the UK. It has been considered an entry level average salary per year of £27,761 as a Biomedical Engineer. And it has been taken the average salary of a Senior Researcher, which is £51,590. Last considerations have been that the total number of hours worked per year are 1,650 and the contribution to the National Insurance is a 12% of gross salary [33].

It must be considered the hardware costs derived from the computer, the electricity consumption costs and establishment costs. As the whole project was made with open-source software, this involves no costs and thus does not appear in the estimate. The linear depreciation is calculated to know the monetary value that the device in question loses every month. Then, it has been calculated the real cost of each product. It is also considered that the real cost is the cost of each device during the time it has been working, the usage time. Knowing that the initial value of the HP Pavilion Laptop 15-ck0xx was 1200€, its estimated residual value, when its useful life has already ended, is 300€, and assuming a product lifetime of 48 months, these are the results:

	USAGE (months)	DEPRECIATION (€/month)	REAL COST (€)	POWER (Wh)
LAPTOP	7	18.75	175	90
MONITOR	7	1.88	14.58	40
DESKTOP COMPUTERS	11	176.25	534.41	200
SUBTOTAL			723.99	

Table 7.2: Hardware costs.

It has been also considered an HP monitor that was used to work as a second screen with an initial value of 120€, an estimated residual value of 30€ and a product lifetime of 48 months. And finally, four desktop computers used by the supervisors. Considering an initial value of £2,500 in average with a 25% of depreciation per year rate and five years expectancy, the residual value for each is of zero. It has been considered that Rachel supervises the project for 7 months and Yo and Daniela for two months. In the table, all the quantities have been expressed in euros.

To calculate the electricity consumption costs, it was searched the power (in watts) of the electronic components used to carry out the project. In 2020, the fixed price of a general consumption rate was 0.1527€/kWh. The power consumption of the laptop was 90Wh and of the screen monitor was 40Wh. And the power consumption of the desktop computers was 200Wh. Therefore, the total electricity consumption cost of the project is 9.60€.

Finally, it has been assumed that the developer and supervisors work from home. The total establishment costs are 5,400€ and considers 300€/month each home office.

A summary table of all the costs analysed is presented to obtain the final cost of the project:

COST TYPE	
LABOUR COSTS	10,810.66€
HARDWARE COSTS	723.99€
SOFTWARE COSTS	0
ELECTRICITY CONSUMPTION COSTS	9.60€
ESTABLISHMENT COSTS	5,400€
TOTAL	**16,944.25€**

Table 7.3: Final cost of the project.

CHAPTER 8. ENVIRONMENTAL IMPACT

The impact in the environment the software may cause is considered negligible as long as it runs in the local client machine and does not require any server. Thus, we can consider its process unnoticeable among the other programs being simultaneously executed.

The development impact can be estimated by summing up the consumption of the main tool, a laptop, and the monitor screen plus the desktop computers. CO_2 emissions were estimated by considering the factor of CO_2 emission in Spain provided by the European Environment Agency, 265.4 g CO_2/kWh [34].

DEVICES	WORKING HOURS	CONSUMPTION	EMISSION ESTIMATE
LAPTOP	442	90 W	10,557.6 g
MONITOR	442	40 W	4,692.3 g
DESKTOP COMPUTERS	27	200 W	1,433.2 g
TOTAL			16,683.1 g CO_2

Table 8.1: Development estimate CO_2 emission.

Finally, the estimated impact of producing the hardware components for the project has also been collected. From the website of the main programmer laptop the CO_2 footprint is estimated to be 200-350 $kgCO_{2e}$ for the laptop and 390-940 $kgCO_{2e}$ for the monitor. From the same website, the estimated impact for a desktop PC is about 300-1500 $kgCO_{2e}$. Considering product lifetime of 48 months, the impact is negligible.

CHAPTER 9. LICENSE

InterMineR-Cytoscape Shiny interface is open-source software, hold by the GNU General Public License (GPL) version 3. This license allows commercial use and modification of the software. The derivations or modified versions can be distributed. Copies of the original software or instructions to obtain copies must be distributed with the software. The software under GPL cannot be under MIT license of BSD type license. In this project, as some of the libraries imported are under GPL license, the software is distributed under GPL.

In the git repository, a file named CREDITS.md contains the license from all the libraries that haven been used in the project. The complete text of the license can also be found in the repository in the file LICENSE.

CHAPTER 10. FUTURE EXTENSIONS

Although InterMineR and Cytoscape were perfectly functional separately, the lack of a user interface between them was making tedious to use them together. InterMineR Cytoscape Shiny interface is the first version of this application. In this section, there are some possible improvements for the following project developers.

Going tab by tab:

- In the Query Builder tab, some paths give an error due to the exceptions in the management of the capital letters (see in the appendix B.8). Some of the exceptions have been captured and treated but some others have not been detected. For this, somebody with Java and JavaScript knowledge could review how the Query Builder works from the web services and do the same in R. Improvements in usability and user-friendliness could be found if more people test the interface.
- In the Run your Query tab, it is desired to increase the height of the table to see more rows which cannot be done directly from R Studio.
- In the Overlay additional data tab, it is desired to find a more efficient way to edit or delete overlays by the user. Currently, the user can visualize the changes in the body's node through a table. In the table are displayed by rows the changes of each node, one by one. The user can select these rows and delete them, removing the changes in the properties and returning to the default value of colour, shape, or size of the body's node. The table enables multiple selections; however, this must be done by clicking each row and cannot be done by click and drag.

Other future lines for the interface are incorporating the InterMine lists and extend the filters.

By including the lists functions of the InterMineR package, developed during the internship, the user could access saved lists from the personal account or predefined lists. If it is desired, in the Visualize your Results tab, the user could save out selected nodes as a list in the personal account.

Additionally, it has been shown necessary to incorporate the possibility of doing operations such as union, intersect, difference, and subtract when applying filters in the Visualize your results and Overlay additional data tab.

The way InterMine queries are constructed means it is sometimes difficult to get all the data into a single query. In the future, it must be explored the possibility to run simultaneous queries or load defined queries for overlay.

Finally, it could be interesting to add the possibility to edit saved networks when they are uploaded in the last tab, Saved Networks.

CONCLUSIONS

The work on the list manager has been merged and released to the Bioconductor version of InterMineR R package.

Regarding to InterMineR Cytoscape interface, after having developed the entire app and tested its behaviour, the result has been appreciated as satisfactory. The application meets all the initial requirements:

- Mines compatibility: All the InterMines are available. In the results, use cases for HumanMine and CovidMine databases have been presented.

- Query Builder: As shown in the use cases, the user can select templates and edit their constraint values or build more complex queries from scratch.

- Interactive and intuitive visualizations: The user can filter the results interacting with a first Cytoscape network. And, in a second tab, the user can edit the colour, shape and size of the nodes according to attributes. Adding more edition tools could be considered as a feature extension.

- Dynamic and reactive user interface: The user can go back and change options from previous tabs.

- Versatility: The application may be run in most desktop environments. This makes InterMineR Cytoscape Interface an universal application.

It is an open source application and so this enables the implementation of new functionalities.

The software development has required from a constant programming and revision of the results. The current situation has led us, my supervisors and myself, to work each one from home. Without the possibility to be face-to-face, a good organisation and agreeing weekly virtual meetings have been keys for the success.

An intensive investment of time has needed to learn object-oriented programming with R but also to do research about Cytoscape and all the R packages used. Nevertheless, the parts with the highest workload have been to communicate the user with the web-services following the HTTP protocol, designing the app to integrate correctly Cytoscape and document all the work done. By the other hand, I have ended the project mastering R Shiny and its complementary packages, and acquiring a lot of experience contributing in an open source big project with a great community of developers.

BIBLIOGRAPHY

[1] RN. Smith, J. Aleksic, D. Butano, A. Carr, S. Contrino, F. Hu, M. Lyne, R. Lyne, A. Kalderimis, K. Rutherford, R. Stepan, J. Sullivan, M. Wakeling, X. Watkins, and G. Micklem. Intermine: a flexible data warehouse system for the integration and analysis of heterogeneous biological data. *Bioinformatics*, 28(23):3163–3165, 2012. 2

[2] S. Ranganathan, M. Gribskov, and K. Nakai. Applications, volume 3. *Encyclopedia of Bioinformatics and Computational Biology*, 3:938–952, 2019. 7

[3] R. Lyne et al. Cross-organism analysis using intermine. *Genesis*, 53(8):547–560, 2015. 7

[4] BioCyc. Guide to The BioCyc Database Collection. https://biocyc.org/biocyc-guide.shtml. Online; accessed 5 June 2020. 7

[5] Michael Cornell, Norman Paton, Shengli Wu, Carole Goble, Crispin Miller, Paul Kirby, Karen Eilbeck, Andy Brass, Andrew Hayes, and Stephen Oliver. Gims - a data warehouse for storage and analysis of genome sequence and functional data. *Bioinformatic and Bioengineering, IEEE International Symposium on*, 0:15, 03 2001. 7

[6] ONDEX. ONDEX Suite Features. http://ondex.sourceforge.net/feats.php. Online; accessed 5 June 2020. 7

[7] GUS/Strategies-WDK. Computational Biology and Informatics Laboratory. http://www.cbil.upenn.edu/node/86. Online; accessed 5 June 2020. 7

[8] Aaron Birkland and Golan Yona. Biozon: A system for unification, management and analysis of heterogeneous biological data. *BMC bioinformatics*, 7:70, 02 2006. 7

[9] T. J. Lee, Yannick Pouliot, Valerie Wagner, Priyanka Gupta, David W. J. Stringer-Calvert, J. D. Tenenbaum, and P. Karp. Biowarehouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7:170 – 170, 2005. 7

[10] S. P. Shah, Xu T. Huang, Y., M. M. Yuen, J. Ling, and B. F Ouellette. Atlas - a data warehouse for integrative bioinformatics. bmc bioinformatics. *BMC Bioinformatics*, 6(34), 2005. 7

[11] Damian Smedley, Syed Haider, Steffen Durinck, and Luca Pandini et al. The BioMart community portal: an innovative alternative to large, centralized data repositories. *Nucleic Acids Research*, 43(W1):W589–W598, 04 2015. 7

[12] Omar S Harb and David S Roos. The eukaryotic pathogen databases: a functional genomic resource integrating data from human and veterinary parasites. *Methods in molecular biology (Clifton, N.J.)*, 1201:1—18, 2015. 7

[13] Thomas Triplet and Gregory Butler. Systems biology warehousing: Challenges and strategies toward effective data integration. 01 2011. 7

[14] Mark Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gaby Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Olavo Bonino da Silva Santos, Philip Bourne, Jildau Bouwman, Anthony Brookes, Tim Clark, Merce Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris Evelo, Richard Finkers, and Barend Mons. The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3, 03 2016. 8

[15] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. https://gephi.org/publications/gephi-bastian-feb09.pdf, 2009. Online; accessed 5 June 2020. 9

[16] David Auber, Daniel Archambault, Romain Bourqui, Antoine Lambert, Morgan Mathiaut, Patrick Mary, Maylis Delest, Jonathan Dubois, and Guy Melançon. The tulip 3 framework: A scalable software library for information visualization applications based on relational data. 01 2012. 9

[17] Andrej Mrvar and Vladimir Batagelj. Analysis and visualization of large networks with program package pajek. *Complex Adaptive Systems Modeling*, 4, 04 2016. 9

[18] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, 30(11):1203–1233, 2000. 9

[19] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003. (software website). 9

[20] Jennifer Chang. Developing an integrated system for biological network exploration. *Graduate Theses and Dissertations*, 15498, 2017. 9

[21] Georgios Pavlopoulos, David Paez Espino, Nikos Kyrpides, and Ioannis Iliopoulos. Empirical comparison of visualization tools for larger-scale network analysis. *Advances in Bioinformatics*, 2017, 07 2017. 10

[22] Ugur Dogrusoz, Alper Karacelik, Ilkin Safarli, Hasan Balci, Leonard Dervishi, and Metin Siper. Efficient methods and readily customizable libraries for managing complexity of large networks. *PLOS ONE*, 13:e0197238, 05 2018. 10

[23] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13(3):57, Sep. 1992. 11

[24] Ensembl. Biomart. https://m.ensembl.org/info/data/biomart/index.html, 2021. Online; accessed 5 June 2020. 11

[25] David Otasek, John Morris, Jorge Bouças, Alexander Pico, and Barry Demchak. Cytoscape automation: Empowering workflow-based network analysis. *Genome Biology*, 20, 09 2019. 12

[26] Julia Gustavsen, Shraddha Pai, Ruth Isserlin, Barry Demchak, and Alexander Pico. Rcy3: Network biology using cytoscape from within r. *F1000Research*, 8:1774, 11 2019. 12

[27] The Human Protein Atlas. Tissue expression of PAX6 - Summary. https://www.proteinatlas.org/ENSG00000007372-PAX6/tissue. Online; accessed 23 March 2021. 33

[28] T M Frayling, J C Evans, M P Bulman, E Pearson, L Allen, K Owen, C Bingham, M Hannemann, M Shepherd, S Ellard, and A T Hattersley. beta-cell genes and diabetes: molecular and clinical characterization of mutations in transcription factors. *Diabetes*, 50(suppl 1):S94, 2001. 38

[29] Lan Shao, Boya Feng, Yuying Zhang, Huanjiao Zhou, Weidong Ji, and Wang Min. The role of adipose-derived inflammatory cytokines in type 1 diabetes. *Adipocyte*, 5(3):270–274, 2016. 38

[30] Soo Lim, Jae Bae, Hyuk-Sang Kwon, and Michael Nauck. Covid-19 and diabetes mellitus: from pathophysiology to clinical management. *Nature Reviews Endocrinology*, 17, 11 2020. 38

[31] Yunzhou Dong, Conrad Fernandes, Yanjun Liu, Yong Wu, Hao Wu, Megan L Brophy, Lin Deng, Kai Song, Aiyun Wen, Scott Wong, Daoguang Yan, Rheal Towner, and Hong Chen. Role of endoplasmic reticulum stress signalling in diabetic endothelial dysfunction and atherosclerosis. *Diabetes and Vascular Disease Research*, 14(1):14–23, 2017. PMID: 27941052. 39

[32] Tobias Boettler and Matthias Herrath. Protection against or triggering of type 1 diabetes? different roles for viral infections. *Expert review of clinical immunology*, 7:45–53, 01 2011. 39

[33] GOV-UK. National Insurance rates and categories. https://www.gov.uk/national-insurance-rates-letters. Online; accessed 19 April 2021. 43

[34] European Environment Agency - EEA. CO2 emission intensity. https://www.eea.europa.eu/data-and-maps/daviz/. Online; accessed 19 April 2021. 45

APPENDICES

TFG TITLE: Development of a GUI for InterMineR and Cytoscape to make biological databases FAIR.

DEGREE: Biomedical Engineering Degree

AUTHOR: Celia Sánchez Laorden

ADVISORS: Dr Rachel Lyne
Dr Gos Micklem

SUPERVISOR: Dr Santiago Marco

DATE: June 13, 2021

APPENDIX A. TASK 1: IMPROVING INTERMINER

A.1. Classes

A.1.1. ListManager-class.R

```r
#' @rdname webservice-class
#' @import S4Vectors
#' @import methods
#' @export

setClass(
  "ListManager",
  representation(
    DEFAULT_LIST_NAME = "character",
    DEFAULT_DESCRIPTION = "character",

    LIST_PATH = "character",
    INTERSECTION_PATH = "character",
    UNION_PATH = "character",
    DIFFERENCE_PATH = "character",
    SUBTRACTION_PATH = "character",
    mine = "character",
    token = "character"
  )
)
```

Listing A.1: ListManager class

A.1.2. webservice-class.R

```r
#' @rdname webservice-class
#' @import S4Vectors
#' @import methods
#' @export

setClass(
  "Service",
  representation(
    mine = "character",
    token = "character"
  )
)
```

Listing A.2: webservice class

A.1.3. InterMineR-class.R

```r
#' @rdname InterMineR-class
#' @import S4Vectors
#' @import methods
#' @export

setClass(
  "InterMineR",
  representation(
    name = "character",
    description = "character",
    select = "character",
    orderBy = "list",
    where = "list"
  )
)
```

Listing A.3: InterMineR class

A.2. Methods and Functions

A.2.1. initInterMine

```
#' @export
##0 - Initilization
# initialize the base and token for future reuse
initInterMine <- function(mine = listMines()["HumanMine"], token=""){
  im <- new("Service",mine = mine, token = token)
  return(im)
}
```

Listing A.4: InterMine.R initInterMine

A.2.2. list_manager

```
#' @rdname webservice-methods
#' @exportMethod list_manager
setGeneric("list_manager", function(object,...){
  standardGeneric("list_manager")
})

#' @rdname webservice-methods
#' @exportMethod list_manager
setMethod(
  "list_manager",
  signature(object = "Service"),
  function(object,...){
    return(new("ListManager",
               DEFAULT_LIST_NAME = 'my_list',
               DEFAULT_DESCRIPTION = 'List created with R client library',

               LIST_PATH = '/lists',
               INTERSECTION_PATH = '/lists/intersect/json',
               UNION_PATH = '/lists/union/json',
               DIFFERENCE_PATH = '/lists/diff/json',
               SUBTRACTION_PATH = '/lists/subtract/json',
               mine = object@mine,
               token = object@token))

})
```

Listing A.5: webservice-methods.R list_manager

A.2.3. ListManager-methods.R

```
#' @name ListManager-methods
#' @import httr

#' @aliases get_list-methods
#' @aliases get_list,ListManager-method
#' @aliases delete_lists-methods
#' @aliases delete_lists,ListManager-method
#' @aliases create_list-methods
#' @aliases create_list,ListManager-method
#' @aliases intersect-methods
#' @aliases intersect,ListManager-method
#' @aliases union-methods
#' @aliases union,ListManager-method
#' @aliases difference-methods
#' @aliases difference,ListManager-method
#' @aliases subtract-methods
#' @aliases subtract,ListManager-method

#GET_api_list: returns the response object of the Request
#' @rdname ListManager-methods
#' @export
setGeneric("GET_api_list", function(object,...){
  standardGeneric("GET_api_list")
})

#' @rdname ListManager-methods
#' @exportMethod GET_api_list
setMethod(
  "GET_api_list",
  signature(object = "ListManager"),
  function(object,...){
    GET(paste0(object@mine, "/service", object@LIST_PATH), add_headers(Authorization = paste("Token", object@token,
         sep = " ")))
  })

```

```r
#' @rdname ListManager-methods
#' @export
setGeneric("get_list", function(object,...){
  standardGeneric("get_list")
})

#' @rdname ListManager-methods
#' @exportMethod get_list
setMethod(
  "get_list",
  signature(object = "ListManager"),
  function(object, list_name){
    resp_list <- GET_api_list(object)
    content_list_parsed <- content(resp_list, "parsed", encoding = "ISO-8859-1")

    exist <- FALSE

    for (list in content_list_parsed$lists){
      if(list$name == list_name){
        return(list)
        exist <- TRUE
      }
    }

    if(exist == FALSE){
      warning(paste0("List", list_name, "doesn't exist."))
      }
    })

#get_unused_list_name: Checks if a list exists by name and it it does it, provides a default name
#' @rdname ListManager-methods
#' @export
setGeneric("get_unused_list_name", function(object,...){
  standardGeneric("get_unused_list_name")
})

#' @rdname ListManager-methods
#' @exportMethod get_unused_list_name
setMethod(
  "get_unused_list_name",
  signature(object = "ListManager"),
  function(object, given_name = 'my_list'){
    resp_list <- GET_api_list(object)
    content_list_parsed <- content(resp_list, "parsed", encoding = "ISO-8859-1")

    list_names <- list()

    for (list in content_list_parsed$lists){
      list_names <- append(list_names, list$name)
    }
    counter <- 1

    name <- object@DEFAULT_LIST_NAME

    if(is.element(given_name, list_names)){

      given_name <- object@DEFAULT_LIST_NAME

      while(is.element(name, list_names)){

        name <- paste0(object@DEFAULT_LIST_NAME, counter)
        given_name <- name
        counter <- counter+1
      }
    }

    return(given_name)
    })

#' @rdname ListManager-methods
#' @export
setGeneric("delete_lists", function(object,...){
  standardGeneric("delete_lists")
})

#' @rdname ListManager-methods
#' @exportMethod delete_lists
setMethod(
  "delete_lists",
  signature(object = "ListManager"),
  function(object, lists){
  #all the names
  resp_list <- GET_api_list(object)
  content_list_parsed <- content(resp_list, "parsed", encoding = "ISO-8859-1")

  all_names <- list()

  for (list in content_list_parsed$lists){
      all_names <- append(all_names, list$name)
    }

  #all the names of template lists

  url2 <- paste0(object@mine, "/service", object@LIST_PATH)
  resp_list2 <- GET(url2)
  content_list_parsed2 <- content(resp_list2, "parsed", encoding = "ISO-8859-1")

  all_names_templates <- list()

  for (list in content_list_parsed2$lists){
    all_names_templates <- append(all_names_templates, list$name)
  }

  for (l in lists){
    name <- l
    if (!(name %in% all_names)){
      warning(sprintf("%s does not exist - skipping", name))
      next
    }
    if (name %in% all_names_templates){
      warning(sprintf("%s is a template list that cannot be deleted", name))
      next
    }
    warning(sprintf("deleting %s", name))

    uri <- paste0(object@mine,"/service", object@LIST_PATH, "?name=", name, "&token=", object@token)

    DELETE(uri, add_headers(Authorization = paste("Token",object@token, sep = " ")))

  }

    #refresh lists/update

  #PATCH(uri,add_headers(Authorization = paste("Token",Token, sep = " "))) Gives status code 501

  #PUT(uri,add_headers(Authorization = paste("Token",Token, sep = " "))) Gives status code 400

  #GET(paste0(object@mine,"/service", '/lists', "?", "&token=", object@token), add_headers(Authorization = paste("
       Token", object@token, sep = " ")))
})

#' @rdname ListManager-methods
#' @export
setGeneric("create_list", function(object,...){
  standardGeneric("create_list")
})

#' @rdname ListManager-methods
#' @exportMethod create_list
setMethod(
  "create_list",
  signature(object = "ListManager"),
  function(object, content, list_type, name = NULL, description = NULL, organism = NULL){

    uri <- paste0(object@mine, "/service/lists?")
    if(is.null(name)){
      name <- get_unused_list_name(object) #this function is created in GET_api_list-get_list-get_unused_list_name.R
    }
    else{
      name <- get_unused_list_name(object, name)
    }

    if(is.null(description)){
      description <- "List created with R Studio client library"
    }

    if(is.list(content)){
      ids <- list()
      for (row in content) {
        ids <- append(ids, row)
      }
      content <- NULL
      for (id in ids) {
        content <- paste(content, id, sep = ",")
      }
      content <- substr(content, 2, nchar(content))
    }
    POST(url = paste0(uri, "name=", name, "&description=", URLencode(description),"&type=", list_type, "&organism=",
         organism),
         body = content, #these are ids
         add_headers(Authorization = paste("Token",object@token, sep = " "),
                     'Content-Type' = "text/plain"))
  })

#do_operation: creates a new list results of an operation, it shouldn't be called directly
#' @rdname ListManager-methods
#' @export
setGeneric("do_operation", function(object,...){
  standardGeneric("do_operation")
})

#' @rdname ListManager-methods
#' @exportMethod do_operation
setMethod(
  "do_operation",
  signature(object = "ListManager"),
  function(object, path, operation, lists, name, description, tags){

    lists_names <- NULL

    for (l in lists){
      lists_names <- paste(lists_names, l, sep = ";")
    }

    lists_names <- substr(lists_names, 2, nchar(lists_names))

    list_names_description <- make_list_names(lists)

    if (is.null(description)){
      description <- sprintf("%s of %s", operation, paste(list_names_description, collapse = " "))
    }

    if (is.null(name)){
      name <- get_unused_list_name(object)
    }else{
      name <- get_unused_list_name(object, name)
    }

    uri <- paste0(object@mine,
                "/service",
                path,
                "?")

    return(POST(paste0(uri, "name=", name, "&lists=", lists_names, "&description=", URLencode(description),"&tags=",
         tags),
                add_headers(Authorization = paste("Token",object@token, sep = " "))))
  })

#make_list_names: turns a list of things into a list of list names
make_list_names <- function(lists){
  list_names <- list()
  for (l in lists){
    try(list_names <- append(list_names, l))
    #maybe more assumptions are needed
  }
  return(list_names)
}

#' @rdname ListManager-methods
#' @export
setGeneric("intersect", function(object,...){
  standardGeneric("intersect")
})

#' @rdname ListManager-methods
#' @exportMethod intersect
setMethod(
  "intersect",
  signature(object = "ListManager"),
  function(object, lists, name = NULL, description = NULL, tags = list()){
    return(do_operation(object, object@INTERSECTION_PATH, "Intersection", lists, name, description, tags))
  })

#' @rdname ListManager-methods
#' @export
setGeneric("union", function(object,...){
  standardGeneric("union")
})

#' @rdname ListManager-methods
#' @exportMethod union
setMethod(
  "union",
  signature(object = "ListManager"),
  function(object, lists, name = NULL, description = NULL, tags = list()){
    return(do_operation(object, object@UNION_PATH, "Union", lists, name, description, tags))
  })

#' @rdname ListManager-methods
#' @export
setGeneric("difference", function(object,...){
  standardGeneric("difference")
})

#' @rdname ListManager-methods
#' @exportMethod difference
setMethod(
  "difference",
  signature(object = "ListManager"),
  function(object, lists, name = NULL, description = NULL, tags = list()){
    return(do_operation(object, object@DIFFERENCE_PATH, "Difference", lists, name, description, tags))
  })

#' @rdname ListManager-methods
#' @export
setGeneric("subtract", function(object,...){
  standardGeneric("subtract")
})

#' @rdname ListManager-methods
#' @exportMethod subtract
setMethod(
  "subtract",
  signature(object = "ListManager"),
  function(object, lefts, rights, name = NULL, description = NULL, tags = list()){

    SUBTRACTION_PATH <- "/lists/subtract/json"

    left_names_description <- make_list_names(lefts)

    right_names_description <- make_list_names(rights)

    left_names<-NULL
    for (l in lefts){
      left_names <- paste(left_names, l, sep = ";")
      }

    left_names <- substr(left_names, 2, nchar(left_names))

    right_names <- NULL

    for (l in rights){
      right_names <- paste(right_names, l, sep = ";")
      }

    right_names <- substr(right_names, 2, nchar(right_names))

    if (is.null(description)){
      description <- sprintf("Subtraction of %s from %s",
                             paste(right_names_description,sep = "and"),
                             paste(left_names_description, sep = "and"))
      }

    if (is.null(name)){
      name <- get_unused_list_name(object)
    }else{
      name <- get_unused_list_name(object,name)
      }

    uri <- paste0(object@mine, "/service", object@SUBTRACTION_PATH, "?")

    return(POST(paste0(uri, "name=", name, "&description=",
                       URLencode(description), "&references=", left_names, "&subtract=", right_names, "&tags=", tags),
                add_headers(Authorization = paste("Token", object@token, sep = " "))))

  })
```

Listing A.6: ListManager-methods.R

A.3. Documentation

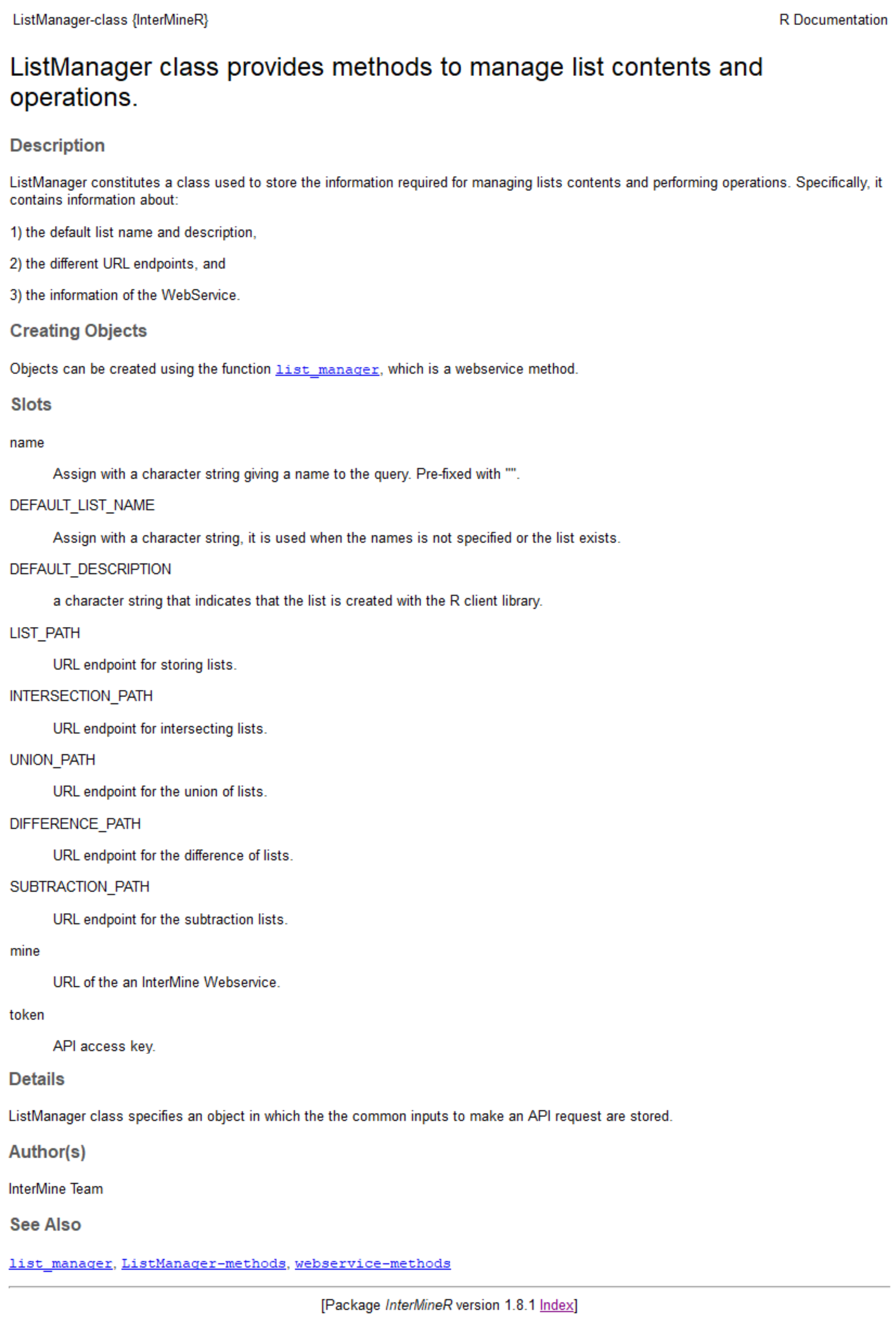
A.3.1. ListManager-class.Rd

```
\docType{class}

\name{ListManager-class}

\alias{ListManager-class}

\title{
ListManager class provides methods to manage list contents and operations.
}

\description{
ListManager constitutes a class used to store the information required for managing lists contents and performing operations. Specifically, it contains information about:

1) the default list name and description,

2) the different URL endpoints, and

3) the information of the WebService.
}

\section{Creating Objects}{
Objects can be created using the function \code{\link{list_manager}}, which is a webservice method.
}

\section{Slots}{
  \describe{
  \item{name}{
Assign with a character string giving a name to the query. Pre-fixed with "".
}
\item{DEFAULT_LIST_NAME}{
Assign with a character string, it is used when the names is not specified or the list exists.
}
\item{DEFAULT_DESCRIPTION}{
a character string that indicates that the list is created with the R client library.
}
\item{LIST_PATH}{
URL endpoint for storing lists.
}
\item{INTERSECTION_PATH}{
URL endpoint for intersecting lists.
}
\item{UNION_PATH}{
URL endpoint for the union of lists.
}
\item{DIFFERENCE_PATH}{
URL endpoint for the difference of lists.
}
\item{SUBTRACTION_PATH}{
URL endpoint for the subtraction lists.
}
\item{mine}{
```

```
URL of the an InterMine Webservice.
}
\item{token}{
API access key.
    }
  }
}

\section{Details}{
ListManager class specifies an object in which the the common inputs to make an API request are stored.
}

\author{
InterMine Team
}

\seealso{
  \code{\link{list_manager}}, \code{\link{ListManager-methods}}, \code{\link{webservice-methods}}
}
```

Listing A.7: Example of Rd file ListManager-class.Rd

A.3.2. ListManager-class documentation rendered to HTML.



ListManager-class {InterMineR} R Documentation

ListManager class provides methods to manage list contents and operations.

Description

ListManager constitutes a class used to store the information required for managing lists contents and performing operations. Specifically, it contains information about:

1) the default list name and description,

2) the different URL endpoints, and

3) the information of the WebService.

Creating Objects

Objects can be created using the function `list_manager`, which is a webservice method.

Slots

name

Assign with a character string giving a name to the query. Pre-fixed with "".

DEFAULT_LIST_NAME

Assign with a character string, it is used when the names is not specified or the list exists.

DEFAULT_DESCRIPTION

a character string that indicates that the list is created with the R client library.

LIST_PATH

URL endpoint for storing lists.

INTERSECTION_PATH

URL endpoint for intersecting lists.

UNION_PATH

URL endpoint for the union of lists.

DIFFERENCE_PATH

URL endpoint for the difference of lists.

SUBTRACTION_PATH

URL endpoint for the subtraction lists.

mine

URL of the an InterMine Webservice.

token

API access key.

Details

ListManager class specifies an object in which the the common inputs to make an API request are stored.

Author(s)

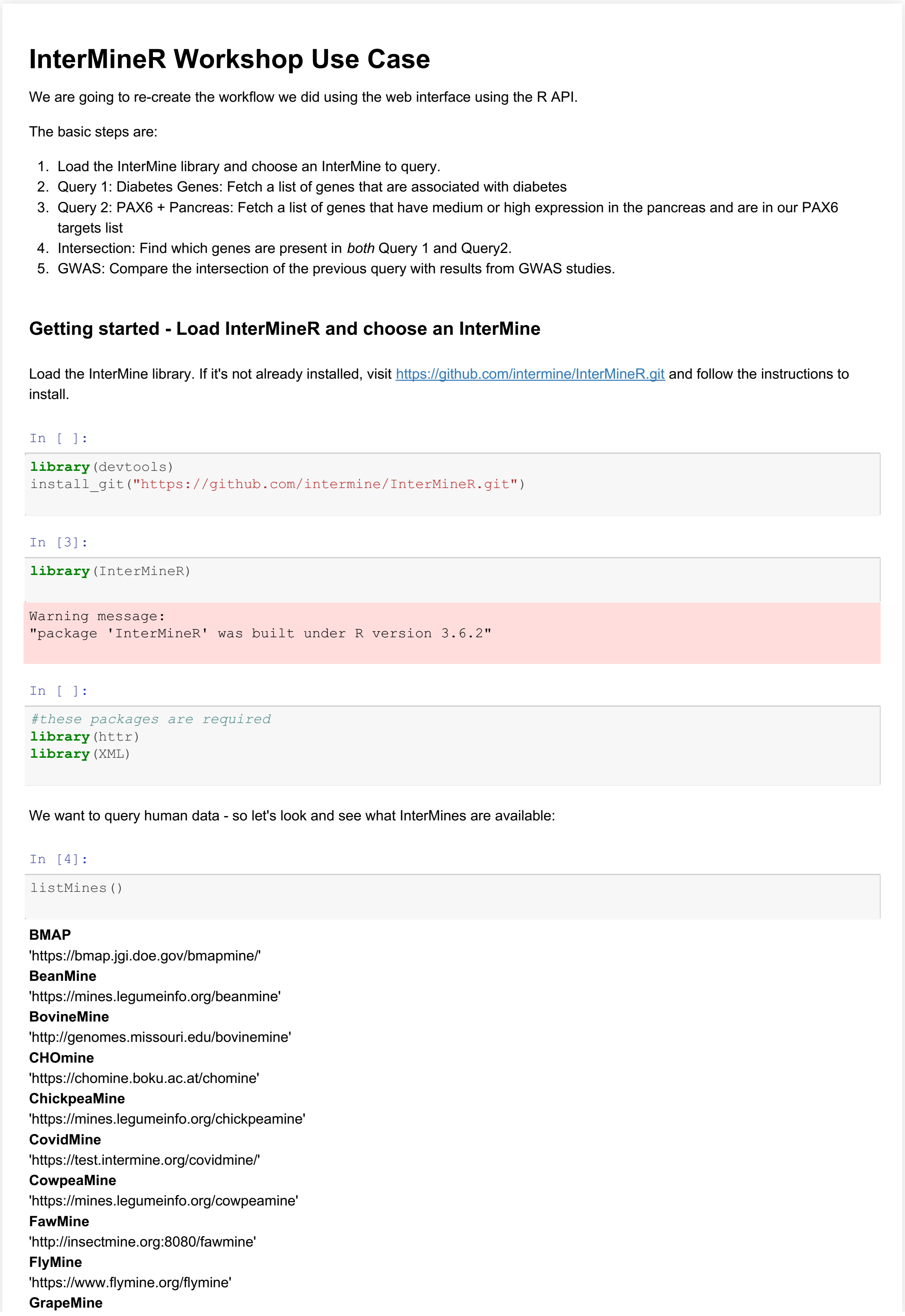
InterMine Team

See Also

`list_manager`, `ListManager-methods`, `webservice-methods`

[Package *InterMineR* version 1.8.1 Index]

A.4. Results



InterMineR Workshop Use Case

We are going to re-create the workflow we did using the web interface using the R API.

The basic steps are:

1. Load the InterMine library and choose an InterMine to query.
2. Query 1: Diabetes Genes: Fetch a list of genes that are associated with diabetes
3. Query 2: PAX6 + Pancreas: Fetch a list of genes that have medium or high expression in the pancreas and are in our PAX6 targets list
4. Intersection: Find which genes are present in *both* Query 1 and Query2.
5. GWAS: Compare the intersection of the previous query with results from GWAS studies.

Getting started - Load InterMineR and choose an InterMine

Load the InterMine library. If it's not already installed, visit https://github.com/intermine/InterMineR.git and follow the instructions to install.

In [ ]:

```
library(devtools)
install_git("https://github.com/intermine/InterMineR.git")
```

In [3]:

```
library(InterMineR)
```

```
Warning message:
"package 'InterMineR' was built under R version 3.6.2"
```

In [ ]:

```
#these packages are required
library(httr)
library(XML)
```

We want to query human data - so let's look and see what InterMines are available:

In [4]:

```
listMines()
```

BMAP
'https://bmap.jgi.doe.gov/bmapmine/'
BeanMine
'https://mines.legumeinfo.org/beanmine'
BovineMine
'http://genomes.missouri.edu/bovinemine'
CHOmine
'https://chomine.boku.ac.at/chomine'
ChickpeaMine
'https://mines.legumeinfo.org/chickpeamine'
CovidMine
'https://test.intermine.org/covidmine/'
CowpeaMine
'https://mines.legumeinfo.org/cowpeamine'
FawMine
'http://insectmine.org:8080/fawmine'
FlyMine
'https://www.flymine.org/flymine'
GrapeMine

'http://urgi.versailles.inra.fr/GrapeMine'
HumanMine
'https://www.humanmine.org/humanmine'
HymenopteraMine
'http://128.206.116.3:8080/hymenopteramine'
IndigoMine
'http://www.cbrc.kaust.edu.sa/indigo'
LegumeMine
'https://mines.legumeinfo.org/legumemine'
LocustMine
'http://locustmine.org:8080/locustmine'
MaizeMine
'http://maizemine.rnet.missouri.edu:8080/maizemine'
MedicMine
'http://medicmine.jcvi.org/medicmine'
MitoMiner
'http://mitominer.mrc-mbu.cam.ac.uk/release-4.0'
ModMine
'http://intermine.modencode.org/release-33'
MouseMine
'http://www.mousemine.org/mousemine'
OakMine
'https://urgi.versailles.inra.fr/OakMine_PM1N'
PeanutMine
'https://mines.legumeinfo.org/peanutmine'
PhytoMine
'https://phytozome.jgi.doe.gov/phytomine/'
PlanMine
'http://planmine.mpi-cbg.de/planmine'
RatMine
'http://ratmine.mcw.edu/ratmine'
RepetDB
'http://urgi.versailles.inra.fr/repetdb'
SoyMine
'https://mines.legumeinfo.org/soymine'
TargetMine
'https://targetmine.mizuguchilab.org/targetmine'
ThaleMine
'https://bar.utoronto.ca/thalemine'
WheatMine
'https://urgi.versailles.inra.fr/WheatMine'
WormMine
'http://intermine.wormbase.org/tools/wormmine/'
XenMine
'http://www.xenmine.org/xenmine'
YeastMine
'https://yeastmine.yeastgenome.org/yeastmine'
ZebrafishMine
'http://zebrafishmine.org'

Okay, let's select HumanMine from the list:

In [5]:

```
humanMine <- listMines()["HumanMine"] #select humanmine
humanMine #print out the value to see what's inside
```

HumanMine: 'https://www.humanmine.org/humanmine'

Okay, now let's tell InterMineR that we want to use HumanMine for our queries. We start by importing the Service class.

Important: you'll need an API token for this part so you can access your HumanMine account. You can get your token by logging into HumanMine and going to the account details tab within MyMine. Cut and paste your token into the code below.

In [6]:

```
Token <- "F16793D0k4BaF5hbe3s0" #insert here your API Acess Key
im <- initInterMine(listMines()["HumanMine"], token = Token)
class(im)
```

First Query: Diabetes Genes

Our first query will be to select all human genes that are associate with diabetes. This will require two constraints:

1. Ensure all genes returned are `Home sapiens` genes (HumanMine contains some non-human genes for homology query purposes)
2. Restrict results to genes that are associated with `diabetes`.

In [7]:

```
query1Diabetes <- setQuery(
  # here we're choosing which columns of data we'd like to see
  select = c("Gene.primaryIdentifier", "Gene.symbol"),
    # set the logic for constraints. The first constraint is the first path+operator+value,
    # e.g. Gene.organism.name = Homo sapiens, and the second constraint is the combination
    # of the second path+operator+value, e.g. Gene.diseases.name CONTAINS diabetes
  where = setConstraints(
    paths = c("Gene.organism.name", "Gene.diseases.name"),
    operators = c("=", "CONTAINS"),
    values = list("Homo sapiens","diabetes")
  )
)
```

Question to ponder: why did we use `=` for our Homo sapiens constraint, but `CONTAINS` for our diabetes constraint?

Anyway, we've set the query up, so now let's actually run it:

In [8]:

```
query1DiabetesResults <- runQuery(im,query1Diabetes)

# and let's print out the first few results to make sure it looks like we'd expect:
head(query1DiabetesResults)
```

Gene.primaryIdentifier	Gene.symbol
1056	CEL
10644	IGF2BP2
11132	CAPN10
1234	CCR5
1493	CTLA4
1636	ACE

We now need to save the list to our intermine account so we can use it again in a later query. The ListManager class provides methods to manage list contents and operations.

In [ ]:

```
im_list <- list_manager(im)
```

To the `create_list` method, we pass the primary identifiers. Before, we delete any list with the same name we want to use for the new list.

In [ ]:

```
ids_query1DiabetesResults <- list(query1DiabetesResults["Gene.primaryIdentifier"])
delete_lists(im_list,c("diabetesGenes"))
create_list(im_list,ids_query1DiabetesResults,list_type="Gene", name="diabetesGenes")
```

In [ ]:

```
print(get_list(im_list,"diabetesGenes"))
```

Query 2: Pax6 targets that have high expression in the Pancreas

This time we're creating another query, but with slightly more complex constraints. We're looking for genes that are in the public HumanMine list `PL_Pax6_Targets`, that are also expressed in the pancreas at a `High` or `Medium` level.

We'll need a few more **constraints** than we did in Query 1:

1. all `Gene`s should be in the list `PL_Pax6_Targets`
2. `Gene.proteinAtlasExpression.tissue.name` should be equal to `Pancreas`
3. `Gene.proteinAtlasExpression.level` should be set to `High` OR `Medium`. This will require two constraints, one for each of medium and high.

We'd also like to see a few more **columns** this time:

1. The `Gene`'s `primaryIdentifier` and `symbol`
2. The following expression data from Protein Atlas:
   - `Gene.proteinAtlasExpression.cellType`
   - `Gene.proteinAtlasExpression.level`
   - `Gene.proteinAtlasExpression.tissue.name`

In [20]:

```
# We don't want to see *all* genes and their expression.
# Let's narrow it down a little by constraining it to genes that are of interest
query2UpInPancreasConstraint = setConstraints(
  paths = c("Gene",
            "Gene.proteinAtlasExpression.level",
            "Gene.proteinAtlasExpression.level",
            "Gene.proteinAtlasExpression.tissue.name"),
  operators = c("IN", rep("=", 3)),
  # each constraint is automatically given a code, allowing us to manipulate the
  # logic for the constraint.
  #  So for us, constraints are set to codes A, B, C, D in order,
  #  e.g. Code A: "Gene" should be "IN" the list named "PL_DiabetesGenes"
  #       Code B: "Gene.proteinAtlasExpression.level" should be equal to "Medium"
  #       Code C: "Gene.proteinAtlasExpression.level" should be equal to "High"
  #       Code D: "Gene.proteinAtlasExpression.tissue.name" should be equal to Pancreas"
  #
  # Now, you might be thinking "how can the expression level be equal to both Medium
  # AND High?" The answer is - it can't, but take a quick look at the constraintLogic
  # we will set in the next code cell for an explanation
  values = list("PL_Pax6_Targets", "Medium", "High", "Pancreas")
)
```

Excellent - we've defined the constraints we want. We still need to choose which columns to view.

In [21]:

```
# Create a new query
query2UpInPancreas = newQuery(
  # Choose which columns of data we'd like to see
  view = c("Gene.primaryIdentifier",
           "Gene.symbol",
           "Gene.proteinAtlasExpression.cellType",
           "Gene.proteinAtlasExpression.level",
           "Gene.proteinAtlasExpression.tissue.name"
  ),
  # set the logic for constraints. This means our pancreas expression level
  # is EITHER Medium (B) or High (C), but not both.
  # --
  # Note: Constraint logic only needs to be set if you wish to use OR. All other
  # constraints have AND logic applied by default.
  constraintLogic = "A and (B or C) and D"
)

# Add the constraint to our expressed pancreas query (previously we just _defined_ the constraint)
query2UpInPancreas$where <- query2UpInPancreasConstraint
```

Remember, that was just setting up the query - we haven't run it yet

In [22]:

```
# Now we have the query set up the way we want, let's actually *run* the query!
query2UpInPancreasResults <-  runQuery(im = im, qry = query2UpInPancreas)

# Show me the first few results please!
head(query2UpInPancreasResults)
```

A data.frame: 6 × 5

Gene.primaryIdentifier	Gene.symbol	Gene.proteinAtlasExpression.cellType	Gene.proteinAtlasExpression.level	Gene.prot
<chr>	**<chr>**	**<chr>**	**<chr>**	**<chr>**
10097	ACTR2	exocrine glandular cells	Medium	Pancreas
10097	ACTR2	islets of Langerhans	Medium	Pancreas
10196	PRMT3	exocrine glandular cells	Medium	Pancreas
10196	PRMT3	islets of Langerhans	Medium	Pancreas
1121	CHM	exocrine glandular cells	Medium	Pancreas
1121	CHM	islets of Langerhans	Medium	Pancreas

Again, we save our results into a list in our account.

In [ ]:

```
ids_query2UpInPancreasResults <- list(query2UpInPancreasResults["Gene.primaryIdentifier"])
delete_lists(im_list,c("UpinPancreas"))
create_list(im_list,ids_query2UpInPancreasResults,list_type="Gene", name="UpinPancreas")
```

In [ ]:

```
print(get_list(im_list,"UpinPancreas"))
```

Intersection: Which genes overlap in Query1 and Query2?

Next, we used a list intersect to find those genes that are upregulated in the pancreas that are also associated with the disease diabetes. We need to intersect the first (UpinPancreas) and second (diabetesGenes) lists that we created. We can do this using the `intersect` method from the ListManager class.

In [ ]:

```
delete_lists(im_list,c("intersectedList"))
print(intersect(im_list,c("UpinPancreas", "diabetesGenes"), "intersectedList"))
```

In [ ]:

```
intersectedList = get_list(im_list,"intersectedList")
print(intersectedList)
```

Here, we can replicate what this method does:

In [23]:

```
# Extract the primaryIdentifier columns from query1 (diabetes genes) and query 2 (upexpressed in p
ancreas)
primaryIdentifiers.diabetes <- query1DiabetesResults[["Gene.primaryIdentifier"]]
primaryIdentifiers.pancreas <- query2UpInPancreasResults[["Gene.primaryIdentifier"]]

# Find the intersection of the two lists of primary identifiers
diabetesAndPancreasGenes <- intersect(primaryIdentifiers.diabetes,primaryIdentifiers.pancreas)

# Show the results
print(diabetesAndPancreasGenes)
```

```
[1] "3172" "6928" "6934"
```

GWAS: Compare the intersection above with results from GWAS studies

Finally, we fed the intersected list from above back into another query to see if there was any association of these genes with diabetes phenotypes according to GWAS studies. Note that we now start our query from the `GWAS` class:

In [24]:

```
# First, we set up the constraints. The last three constraints are the
# diabetesAndPancreas result genes from our last query.
query3GWASConstraints <- setConstraints(
    paths = c("GWAS.results.pValue",
              "GWAS.results.phenotype",
              # using rep so we don't have to type this three times...
              rep("GWAS.results.associatedGenes.primaryIdentifier",3)
             ),
    operators = c("<=",
                  "CONTAINS",
                  rep("=",3)),
    values = list("1e-04",    #A
                  "diabetes",#B
                  "3172",     #C
                  "6928",     #D
                  "6934")     #E
  )
```

Now we've set our constraints up nicely, let's choose which columns we want to view.

In [25]:

```
query3GWAS <- newQuery(
  # Quite a few columns this time!
  view = c("GWAS.results.associatedGenes.primaryIdentifier",
    "GWAS.results.associatedGenes.symbol", "GWAS.results.associatedGenes.name",
    "GWAS.results.SNP.primaryIdentifier", "GWAS.results.pValue", "GWAS.results.phenotype",
    "GWAS.firstAuthor", "GWAS.name", "GWAS.publication.pubMedId",
    "GWAS.results.associatedGenes.organism.shortName"),
    # set the logic for constraints. Remember that we want our results
    # to include any one of the three genes we found in the list of diabetes+pancreas genes
    # so we need to use some OR logic.
  constraintLogic = "A and B and (C or D or E)"
)
```

Add the constraints to the query, and then run it...

In [27]:

```
#add constraint
query3GWAS$where <- query3GWASConstraints
#run query
query3GWASResults <- runQuery(im, query3GWAS)
```

Now, let's view those results...

In [30]:

```
query3GWASResults
```

A data.frame: 46 × 10

GWAS.results.associatedGenes.primaryIdentifier	GWAS.results.associatedGenes.symbol	GWAS.results.associatedGenes.r
<chr>	<chr>	<chr>
3172	HNF4A	hepatocyte nuclear factor 4 alpha
3172	HNF4A	hepatocyte nuclear factor 4 alpha
3172	HNF4A	hepatocyte nuclear factor 4 alpha
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2

GWAS.results.associatedGenes.primaryIdentifier	GWAS.results.associatedGenes.symbol	GWAS.results.associatedGenes.r
<chr>	<chr>	<chr>
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2

GWAS.results.associatedGenes.primaryIdentifier	GWAS.results.associatedGenes.symbol	GWAS.results.associatedGenes.
<chr>	<chr>	<chr>
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2

GWAS.results.associatedGenes.primaryIdentifier	GWAS.results.associatedGenes.symbol	GWAS.results.associatedGenes.r
6934 **<em>**	TCF7L2 **<em>**	transcription factor 7 like 2 **<em>**
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2

GWAS.results.associatedGenes.primaryIdentifier	GWAS.results.associatedGenes.symbol	GWAS.results.associatedGenes.r
<chr>	<chr>	<chr>
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2

GWAS.results.associatedGenes.primaryIdentifier	GWAS.results.associatedGenes.symbol	GWAS.results.associatedGenes.r
<chr>	<chr>	<chr>
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2
6934	TCF7L2	transcription factor 7 like 2

GWAS.results.associatedGenes.primaryIdentifier	GWAS.results.associatedGenes.symbol	GWAS.results.associatedGenes.
<chr>	<chr>	<chr>
6934	TCF7L2	transcription factor 7 like 2

And let's take a look at the unique gene symbols that were returned:

In [33]:

```
GWASIds <- query3GWASResults["GWAS.results.associatedGenes.symbol"]
unique(GWASIds)
```

A data.frame: 2 × 1

	GWAS.results.associatedGenes.symbol
	<chr>
1	HNF4A
4	TCF7L2

APPENDIX B. TASK 2: SHINY INTERFACE

B.1. Welcoming message

```html
<p><img style="display: block; margin-left: auto; margin-right: auto;" src="logo.png" width="100" height="100"/></p>
<h1 style="text-align: center;">WELCOME TO&nbsp;<strong>InterMineR Cytoscape Interface</strong></h1>
<h4 style="text-align: center;">An interactive open source software for the integration and analysis of InterMine data warehouse getting the most out of Cytoscape visualizations.</h4>
<hr />
<p style="text-align: left;">This interface wants to be a guide to run queries and interpret them with the intuitive Cytoscape visualizations without prior software experience. It facilitates understanding and communication of relevant relationships between different biological Data Classes. </p>
<hr />
<p style="text-align: left;">&nbsp;<strong>WHAT YOU CAN DO WITH IT:</strong></p>
<ul>
<li style="text-align: left;">Run queries using whatever <strong><span style="color: #37D624;">template</span>&nbsp;</strong> you want from all registered InterMine instances in one place.&nbsp;</li>
<li style="text-align: left;">Advanced users can use a <strong><span style="color: #37D624;">flexible query</span>&nbsp;</strong> interface to construct their own data mining queries.&nbsp;</li>
<li style="text-align: left;">Display and <strong><span style="color: #37D624;">export</span>&nbsp;</strong> the <strong><span style="color: #37D624;">results</span>&nbsp;</strong> in a table.&nbsp;</li>
<li style="text-align: left;">A set of <strong><span style="color: #37D624;">visualization tools</span>&nbsp;</strong> are made available to the user from Cytoscape domains to enrich the interpretation of the results.&nbsp;</li>
<li style="text-align: left;">And <strong><span style="color: #37D624;">customization</span>&nbsp;</strong> options to personalize the Cytoscape Networks.&nbsp;</li>
<li style="text-align: left;"><strong><span style="color: #37D624;">Store</span>&nbsp;</strong> your visualizations in <strong><span style="color: #37D624;">JSON</span>&nbsp;</strong> format and display <strong><span style="color: #37D624;">saved Networks</span>&nbsp;</strong>.&nbsp;</li>
</ul>
<hr />
<p style="text-align: left;"><span style="display: inline;">If you need any help click the <strong>?</strong>&nbsp;in each of the pages. Some hints are also visible hovering your mouse over the buttons.&nbsp;</span></p>
<p style="text-align: center;"><span style="color: #000000;">InterMine Project - University of Cambridge, 2020 | c</span><span style="color: #000000;">ontact: csanchla8@alumnes.ub.edu</span></p>
```

Listing B.1: Script of the intro_text.html.

B.2. Interface simplified code

B.2.1. Simplified code of the app.R script structure

```r
## app.R ##
#Packages----
# load("Packages.R")
#Libraries----

    ...

#Load functions----

    ...

# UI: View----
#user interface should contain what the users see
ui <- dashboardPage(

  # title of browser tab
  title = "Data Visualizations with RCytoscape",

  dashboardHeader(
    title = tags$img(src = "intermine.png", width = "100%"),
    dropdownMenuOutput("dropdownmenu")
  ),

  dashboardSidebar(

    sidebarMenu(

    ...

    )
  ),

  dashboardBody(
    # Add class fixed to header and sidebar
    tags$script(HTML("$('body').addClass('fixed');")),

    tabItems( # Create a tab items menu

      tabItem( # Create a new tab
        tabName = "home",
        includeMarkdown("home.md")
      ),

        ...
    )
  )
)

#Server: where the data is processed-----
server <- function(input, output, session){
  # Close background process when user close app
  if (!interactive()) {
    session$onSessionEnded(function() {
      stopApp()
      q("no")
    })
  }

    ...

}
# Call ui and server to create a new session in browser which display the app----
shinyApp(ui, server, options = list(launch.browser = TRUE))
```

Listing B.2: Simplified code of the app.R script structure.

B.2.2. Simplified code of the tab dashboard structure of:

B.2.2.1. Run your query and Saved Results tabs

```r
# UI: View----
#user interface should contain what the users see
ui <- dashboardPage(

    ...

  dashboardBody(

    ...

    tabItems(

        ...

      tabItem(
        tabName = "download", #label for Saved Results tab
        introjsUI(), #to use the package rintrojs
        tags$h3("Saved Workflows"), #header text
        tabPanel("Cytoscape Network Style", fluid = TRUE,
                 fluidRow( #fluid row section
                   box( #box to hold content in the fluid row section
                     width = 12,
                     column(
                       width = 12,
                       fileInput("file", "Upload Zip file", accept = ".zip"), #file upload control
                       bsTooltip("file", "Here, you can upload saved networks and see the results of the query in a
                            table and the interactive Network with your last modifications."),
                       actionButton("unzip", "Unzip files", style="color: #fff; background-color: #3366ff; border-
                            color: #3366ff"),
                       bsTooltip("unzip", "Press this button to display the results table and the network."),
                       br(), # add a line break
                       br(),
                       textOutput("zipError") # Output error
                     )

                   ),
                   conditionalPanel("input.unzip",{ #only if the unzip is correct
                     box(
                       width = 12,
                       column(
                         width = 12,
                         withLoader(DTOutput("resultstable_save", height = "100%"), loader = "loader6"),
                         br(),
                         hr(),
                         cytoscapeOutput("network_saved", height = "600px")
                       )
                     )
                   })
                 )

        ))
    )
  )
)
```

Listing B.3: Code for the tab dashboard structure of Saved Results tab.

B.2.2.2. The other tabs

```r
# UI: View----
#user interface should contain what the users see
ui <- dashboardPage(

    ...

  dashboardBody(

    ...

    tabItems(

        ...

      tabItem( # add new item inside tab files
        tabName = "templates",

        introjsUI(), #include rintrojs

        useShinyFeedback(), # include shinyFeedback

        tags$h3("Template queries"),
        tabPanel("Template queries", fluid = TRUE,

                 sidebarLayout(
                   sidebarPanel( #display with distinc background color and containing input controls
                     fluidRow(
                       column( # create a column
                         width = 12, # fill width

                         uiOutput("template_mine"), #output text for the user saying which Mine is chosen
                         hr(),
                         uiOutput("template_choice"), #the select list to choose a template query, the values are defined once the Mine is selected
                         br(),
                         br(),
                         br(),
                         #include actionable button to display help information
                         actionButton("help1","", icon = icon("question"), style="color: #000; background-color: #bdff80; border-color: #bdff80")

                       ))),
                   mainPanel( #ocupies approx 2/3 of the width, usually contains outputs
                     fluidRow(
                       uiOutput("index_choice"),
                       hr(),
                       br(),
                       uiOutput("template_constraint_summary"), #control inputs to redefine constraints from the template query
                       bsTooltip("template_constraint_summary","Summarize the information about the constraints contained by an object of the class InterMineR.","top")
                     )
                   )
                 )


        )

      ),

        ...

    )
  )
)
```

Listing B.4: Code for the tab dashboard structure of Template Queries tab.

B.3. InterMineR fragments of code

B.3.1. Selecting the InterMine and getting the data model: initInterMine, listMines, getModel

```
#Server: where the data is processed-----
server <- function(input, output, session){

    ...

  ###### Choosing the Mine ######
  im <- ""
  #functions from the InterMineR package are called
  im <- reactive({
    initInterMine(mine = listMines()[input$mine_template]) #saves the mine
    })
  model_im <- reactive({
    getModel(initInterMine(mine = listMines()[input$mine_template]))
    })

    ...

}
```

Listing B.5: Code for choosing the InterMine.

B.3.2. Get the information of the templates pre-defined in InterMine and get the query obtained in a template: getTemplates, getTemplatesQuery

```
#Server: where the data is processed-----
server <- function(input, output, session){

    ...

```

```r
  ###### Choosing the Template ######
  output$template_choice <- renderUI({
    selectInput(
      inputId = "t_choice",
      label = "Choose a template query:",
      choices = getTemplates(im())[2], #function from InterMineR package that returns a list of templates
      selected = ""
    )

  })

    ...

  q_reactive <- eventReactive(input$t_choice,{
    #getting the query contained in a template using a function from the InterMineR package
    getTemplateQuery(im(), getTemplates(im())[1][[1]][match(input$t_choice,getTemplates(im())[2][[1]])])
  })

    ...

}
```

Listing B.6: Code for choosing the template query.

B.3.3. Constraints: setConstraints

```r
#Server: where the data is processed-----
server <- function(input, output, session){

    ...

  ###### Set the Query Template ######
  #the following return the results from the (non)costumized template query in response to the events
  q_query_reactive <- eventReactive(c(input$t_choice,input$m.index_t1,input$m.index_t2,input$m.index_t3,input$m.index_
       t4,input$values_t1,input$values_t2,input$values_t3, input$values_t4),{ #these are the events inside a vector
        q <- q_reactive() #getting the query contained in a template
        ind_1 <- input$m.index_t1 #path of the constraint of the template (information by default)
        value_1 <- input$values_t1 #argument of the constraint of the template (can be edited by the user)
        if(!(is.null(value_1))){ #the user has edit the argument of the constraint
          q_constraints <- setConstraints( #InterMineR package function to modify constraints
            values = list(c(input$values_t1)),
            modifyQueryConstraints = q, #the template
            m.index = 1 #first constraint and so index 1
          )
          #same operation for the other constraints
          ind_2 <- input$m.index_t2
          value_2 <- input$values_t2
          if(value_2!=""){
            q_constraints <- setConstraints(
              values = list(value_1, value_2),
              modifyQueryConstraints = q,
              m.index = c(1,2)
            )
            ind_3 <- input$m.index_t3
            value_3 <- input$values_t3
            if(value_3!=""){
              q_constraints <- setConstraints(
                values = list(value_1, value_2, value_3),
                modifyQueryConstraints = q,
                m.index = c(1,2,3)
              )
              ind_4 <- input$m.index_t4
              value_4 <- input$values_t4
              if(value_4!=""){
                q_constraints <- setConstraints(
                  values = list(value_1, value_2, value_3, value_4), #different constraints and so different values
                  modifyQueryConstraints = q, #the template
                  m.index = c(1,2,3,4) #the index of the paths from the template
                )
              }
            }
          }

          q_query <- setQuery(inheritQuery = q, where = q_constraints) #combining the query template with the new
               constraints
        }
        else{
          q_query <- q #or in case there are no modifications, the template itselves
        }
    })

    ...

}
```

Listing B.7: Code for modifying the constraints of a Query Template.

```
#Server: where the data is processed-----
server <- function(input, output, session){

    ...

  ###### Setting the Query from the Builder Query ######
  #Shown data
  q_query_reactive_builder_select <- eventReactive(c(input$b_choice,input$b_2_choice,input$b_3_choice,
                                                     input$b_4_choice,input$b_5_choice,input$b_select_1,
                                                     input$b_constraint_1,input$b_order,input$desc_asc,
                                                     input$b_select_2, input$b_constraint_2,input$b_select_3,
                                                     input$b_constraint_3, input$b_select_4,input$b_constraint_4,
                                                     input$b_select_5, input$b_constraint_5,input$operator_0,
                                                     input$value_0, input$operator_1, input$value_1,input$operator_2,
                                                     input$value_2, input$operator_3, input$value_3,input$operator_4,
                                                     input$value_4, input$operator_5, input$value_5),{

       model_mine <- model_im() #getModel representation of the data model for the mine
       selectitems <- c() #this is going to be the first argument of setQuery
       #type of data to be returned in the first level
       if(!(is.null(input$b_select_1))){ #data "class" shown in the 1st level
         try({
           for (element in input$b_select_1){
             element_str <- strsplit(element, " ") #creating a list of words
             element_str <- element_str[[1]]
             try({
               for (variable in element_str[2:length(element_str)]) {
                 #trying to catch the words in capital letters that are not SNP
                 #future improvement: catch more exceptions such as SNP...
                 #it has been observed while building a query that some classes are not recognized due to the writting
                 if(variable==toupper(variable) & variable!="SNP"){
                   #the modification: capitalize (only the first letter) the variables in capital letters
                   element_str <- replace(element_str, element_str==variable,capitalize(tolower(variable)))
                 }
               }
             },silent = TRUE)

             if(element_str==toupper(element_str)){
               element_str[1] <- tolower(element_str[1]) #the first entire word to lower
             }else{
               element_str[1] <- decapitalize(element_str[1]) #the first letter of the first word to lower
             }
             element_str <- paste(element_str, collapse = "") #paste again all the splitted words together
             selectitems <- c(selectitems,paste0(input$b_choice,".",element_str)) #add path
           }
         })
       }
       #the same for the consecutive levels
        ...
       selectitems #returns
    })

  #Order
  q_query_reactive_builder_order <- eventReactive(c(input$b_choice,input$b_order,input$desc_asc),{
    #first split, decapitalize, paste again
    b_order_str <- strsplit(input$b_order, " ")
    b_order_str[[1]][1] <- decapitalize(b_order_str[[1]][1])
    b_order_str <- paste(b_order_str[[1]], collapse = "")

    order <- paste0(input$b_choice,".",b_order_str)
    sort <- c(as.character(input$desc_asc))
    names(sort) <- order
    return(list(sort)) #list, the name of the column and the type of sorting

  })
  #Constraint data
  q_query_reactive_builder_constraints <- eventReactive(c(input$b_choice,input$b_2_choice,input$b_3_choice,
                                                     input$b_4_choice,input$b_5_choice,input$b_select_1,
                                                     input$b_constraint_1,input$b_order,input$desc_asc,
                                                     input$b_select_2, input$b_constraint_2,input$b_select_3,
                                                     input$b_constraint_3, input$b_select_4,input$b_constraint_4,
                                                     input$b_select_5, input$b_constraint_5,input$operator_0,
                                                     input$value_0, input$operator_1, input$value_1,input$operator_2,
                                                     input$value_2, input$operator_3, input$value_3,input$operator_4,
                                                     input$value_4, input$operator_5, input$value_5),{
        model_mine <- model_im()
        constraints_paths <- c() #this is going to be the first argument of setConstraints()
        constraints_operators <- c() #this is going to be the second argument of setConstraints()
        constraints_values <- c() #this is going to be the third argument of setConstraints()

        #consider that from the first data type level there are two input controls for constraints
        if(!(is.null(input$operator_0)) & !(is.null(input$value_0))){
          try({
            constraints_paths <- c(constraints_paths,input$b_choice)
            constraints_operators <- c(constraints_operators,input$operator_0)
            #if more than one value corresponds to a single constraint, they create a list
            constraints_values <- rlist::list.append(constraints_values, str_split(input$value_0,","))
          })
        }
        #treatment as with q_query_reactive_builder_select to define the paths for the query
        if(!(is.null(input$operator_1)) & !(is.null(input$value_1))){
          try({
            b_select_1 <- strsplit(input$b_constraint_1, " ")
            if(b_select_1==toupper(b_select_1)){
              b_select_1[[1]][1] <- tolower(b_select_1[[1]][1])
            }else{
              b_select_1[[1]][1] <- decapitalize(b_select_1[[1]][1])
            }
            b_select_1_str<-paste(b_select_1[[1]], collapse = "")

            constraints_paths <- c(constraints_paths,paste0(input$b_choice,".",b_select_1_str))
            constraints_operators <- c(constraints_operators,input$operator_1)
            constraints_values <- rlist::list.append(constraints_values, str_split(input$value_1,","))
          })
        }
        #the same for the consecutive constraints and levels

            ...

        constraints_values_list <- list() #order list
        index<-1
        for(i in constraints_values){
          constraints_values_list[index]<-i
          index<-index+1
        }

        constraint_function <- setConstraints(
          paths = constraints_paths, #list with the path(s)
          operators = constraints_operators, #operator(s) for each constraint
          values = constraints_values_list #order
        )

        constraint_function #return
     })

    ...

}
```

Listing B.8: Code for defining the type of data to be returned and the constraints of a new query.

B.3.4. Initialize a new InterMineR query or modify an existing list query: setQuery

```
#Server: where the data is processed-----
server <- function(input, output, session){

    ...

  ###### Setting the Query from the Builder Query ######

    ...

  #setQuery
  q_query_reactive_builder <- eventReactive(c(input$setQ, input$delete_constraints),{
    values_select$dfSelect <- q_query_reactive_builder_select() #the data frame with type of data to be returned by
         the query
    if (!is.null(input$table_constraints_rows_selected)) { #double check of selected paths with the table (summmary)
      #the user could had deleted some paths
      values_select$dfSelect <- data.frame(values_select$dfSelect)
      values_select$dfSelect <- values_select$dfSelect[-as.numeric(input$table_constraints_rows_selected),]
    }
    #when is dataframe the information is inside [,1]
    if(is.data.frame(values_select$dfSelect)){
      values_select <- values_select$dfSelect[,1]
    }else{
      values_select <- values_select$dfSelect
    }
    #using the function from intermineR package
    q_query_reactive_builder_func <- setQuery(
      select = values_select, #type of data to be returned
      orderBy = q_query_reactive_builder_order(), #order of the results
      where = q_query_reactive_builder_constraints() #constraints
    )
    q_query_reactive_builder_func #return
  })

    ...

}
```

Listing B.9: Code for creating a query.

B.3.5. Get the summary of constraints: summary

```
#Server: where the data is processed-----
server <- function(input, output, session){

    ...
```

```

  ###### Set the Query Template ######

    ...

  #the following returns a summary of the (non)modified template query in response to the events
  observeEvent(c(input$t_choice,input$values_t1,input$m.index_t2,
                 input$m.index_t3,input$m.index_t4,
                 input$values_t1,input$values_t2,input$values_t3,
                 input$values_t4),{
                   output$template_constraint_summary <- renderUI({
                     tagList(
                       box(
                         width = 12, # fill the width of the page
                         tags$h2("Summary"),
                         #only showing query information with constraints
                         if(is.list(q_query_reactive())){tags$p("No constraint set to template query.")}else{try(
                               renderDataTable({summary(q_query_reactive())}))},
                         br(),
                         div(style="display:inline-block; float:right",
                             actionButton("goResults","Go to Results",
                                          style="color: #fff; background-color: #3366ff; border-color: #3366ff"))
                       )
                     )
                   })
                 })
    ...

  ###### Setting the Query from the Builder Query ######

    ...

  #Summary of the query
  observeEvent(c(input$setQ,input$delete_constraints),{
    output$builder_constraint_summary <- renderUI({
      tagList(
        box(
          width = 12, # fill the width of the page
          tags$h2("Summary"),
          if(is.list(q_query_reactive_builder())){tags$p("No constraint set to template query.")}else{
            if(length(summary(q_query_reactive_builder())) == 1){try(tags$p(summary(q_query_reactive_builder())),
                  silent=TRUE
            )}else{try(datatable(summary(q_query_reactive_builder())),silent=TRUE
            )}
          },
          div(style="display:inline-block; float:right",
              actionButton("goBuilder","Go to Results",
                           style="color: #fff; background-color: #3366ff; border-color: #3366ff"))
        )
      )
    })
  })

    ...

}
```

Listing B.10: Code for the summary of constraints.

B.3.6. Get the results: runQuery

```
#Server: where the data is processed-----
server <- function(input, output, session){

    ...

  ###### Run Template Query ######
  results_reactive <- eventReactive(c(input$t_choice,input$m.index_t1,input$m.index_t2,
                                      input$m.index_t3,input$m.index_t4,
                                      input$values_t1,input$values_t2,input$values_t3,
                                      input$values_t4,input$goResults),{ #the more advanced in the workflow the more
                                            events
                                        #reactivity is that when a parameter of the template query is changed the
                                              results are obtained again
                                        q <- q_query_reactive()
                                        if(is.list(q)){
                                          res <- runQuery(im(), q)
                                        } else {
                                          res <- runQuery(im(), q)
                                        }
                                      })
    ...

  ###### Setting the Query from the Builder Query ######

    ...

  #RunQuery
  results_reactive_builder <- eventReactive(c(input$goBuilder, input$delete_constraints),{
    q <- q_query_reactive_builder()
    if(is.list(q)){
      res <- runQuery(im(), q[[1]])
    } else {
      res <- runQuery(im(), q)
    }
  })

    ...

}
```

Listing B.11: Code that returns results from a query.

B.4. Cytoscape fragments of code

B.4.1. Visualize tab

B.4.1.1. From data frame results to JSON graph: igraph and graphNEL

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Selection of Id and Source ######
    ...
  observeEvent(input$options_button,{ #the attributes selected in this step are the parameters to play with the
       network visualization
    if(identical(modality(),NULL)){
      updateSelectInput(session, "nodes_attributes",
                        choices = names(results_reactive()))
    }else{
      updateSelectInput(session, "nodes_attributes",
                        choices = names(results_reactive_builder()))
    }

  })
  observeEvent(input$options_button,{
    if(identical(modality(),NULL)){
      updateSelectInput(session, "id_edges",
                        choices = names(results_reactive()))
    }else{
      updateSelectInput(session, "id_edges",
                        choices = names(results_reactive_builder()))
    }

  })
  observeEvent(input$options_button,{
    if(identical(modality(),NULL)){
      updateSelectInput(session, "edges_attributes",
                        choices = names(results_reactive()))
    }else{
      updateSelectInput(session, "edges_attributes",
                        choices = names(results_reactive_builder()))
    }

  })

  ###### Dataframe of Results converted to graphNEL class ######
  interaction_reactive_func <- function(dataframe){
    df <- as.data.frame(dataframe) #results_reactive() or results_reactive_builder()

    nodes <- data.frame(id = unique(c(df[,input$id_nodes], df[,input$id_edges])),
                        stringsAsFactors = FALSE)

    edges <- df %>%
      dplyr::select(source = input$id_nodes,
                    target = input$id_edges) %>%
      dplyr::mutate(interaction = paste(source, '_', target))

    i_graph <- graph_from_data_frame(edges, directed = TRUE, nodes) #conversion to igraph

    for(element in input$edges_attributes){
      edge_attr(i_graph, element) <- df[,element]
    }
    for(element in input$nodes_attributes){
      df <- df[!duplicated(df[,input$id_nodes]),] #duplicated id nodes are eliminated
      vertex_attr(i_graph, element) <- df[,element]
    }

    g <- igraph.to.graphNEL(i_graph) #conversion to graphNEL class

  }
  interaction_reactive <- eventReactive(c(input$t_choice,input$m.index_t1,input$m.index_t2,
                                          input$m.index_t3,input$m.index_t4,
                                          input$values_t1,input$values_t2,input$values_t3,
                                          input$values_t4,input$goResults,input$goInteraction, input$id_nodes,
                                          input$id_edges,input$nodes_attributes,input$edges_attributes),{

                                            interaction_reactive_func(results_reactive())
                                          })
  interaction_reactive_builder <- eventReactive(c(input$goInteraction,input$id_nodes,input$id_edges,
                                                 input$nodes_attributes,input$edges_attributes),{

                                                   interaction_reactive_func(results_reactive_builder())
                                                 })
    ...
}
```

Listing B.12: Code that converts results to a graphNEL class.

B.4.1.2. cyjShiny and options

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Options of the Visualize your results tab ######
  observeEvent(input$goInteraction,{
    if(identical(modality(),NULL)){
      updateSelectInput(session, "selectName", #Select Node by ID
                        choices = c("",nodes(interaction_reactive()))) #retrieves the nodes ID of the graphNEL
    }else{
      updateSelectInput(session, "selectName",
                        choices = c("",nodes(interaction_reactive_builder())))
    }
  })

  nodes_attr_reactive <- reactive({ #to read as a reactive variable
    input$nodes_attributes
  })
  observeEvent(input$goInteraction, {
    if(identical(modality(),NULL)){
      updateSelectInput(session, "selectName_2", #select node by attribute
                        choices = c("",nodes_attr_reactive()))
    }else{
      updateSelectInput(session, "selectName_2",
                        choices = c("",nodes_attr_reactive()))
    }

  })

  observeEvent(input$selectName_2, ignoreInit = TRUE, { #once the attribute is set, choose the value
    if(identical(modality(),NULL)){
      df <- results_reactive()
      updateSelectInput(session, "selectName_2_attr",
                        choices = c("",df[,input$selectName_2]))
    }else{
      df <- results_reactive_builder()
      updateSelectInput(session, "selectName_2_attr",
                        choices = c("",df[,input$selectName_2]))
    }
  })
    ...
}
```

Listing B.13: Code that establishes the susceptible data to be filtered.

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Options of the Visualize your results tab ######
    ...

  observeEvent(input$selectName,  ignoreInit=TRUE,{
    printf("about to sendCustomMessage, selectNodes")
    session$sendCustomMessage(type="selectNodes", message=list(input$selectName))
    #using selectNodes function from cyjShiny in the form of a custom message to the web page
    #to select nodes by ID in the network
  })

  observeEvent(input$selectName_2_attr,  ignoreInit=FALSE,{
    if(identical(modality(),NULL)){
      printf("about to sendCustomMessage, selectNodes")
      df <- results_reactive()

      node_i <- df[,input$selectName_2]==input$selectName_2_attr #the nodes that have the attribute
      node <- df[node_i,input$id_nodes] #the node ID is got

      for (element in node){ #iterate because the function selectNodes only accepts one node
        session$sendCustomMessage(type="selectNodes", message=list(element))
        #selecting nodes by attribute value
      }
    }else{
      printf("about to sendCustomMessage, selectNodes")
      df <- results_reactive_builder()

      node_i <- df[,input$selectName_2]==input$selectName_2_attr
      node <- df[node_i,input$id_nodes]

      for (element in node){
        session$sendCustomMessage(type="selectNodes", message=list(element))
      }
    }
  })

  observeEvent(input$sfn,  ignoreInit=TRUE,{
    printf("about to sendCustomMessage, sfn")
    #select the first neighbors
    session$sendCustomMessage(type="sfn", message=list())
  })

  observeEvent(input$fit, ignoreInit=TRUE, {
    fit(session, 80) #pixels
  })

  observeEvent(input$fitSelected,  ignoreInit=TRUE,{
    printf("about to call R function fitSelected")
    #the current selected nodes fill the display
    fitSelected(session, 100)
  })

  observeEvent(input$getSelectedNodes, ignoreInit=TRUE, {
    output$selectedNodesDisplay <- renderText({" "})
    getSelectedNodes(session)
    #print the ID's of the node selection
  })

  observeEvent(input$selectedNodes, {

    #  communicated here via assignement in cyjShiny.js
    #     Shiny.setInputValue("selectedNodes", value, {priority: "event"});
    newNodes <- input$selectedNodes;
    output$selectedNodesDisplay <- renderText({
      paste(newNodes)
    })
  })

  observeEvent(input$clearSelection,  ignoreInit=TRUE, {
    printf("about to sendCustomMessage, clearSelection")
    session$sendCustomMessage(type="clearSelection", message=list())
  })

  observeEvent(input$doLayout,  ignoreInit=TRUE,{
    strategy <- input$doLayout
    printf("about to sendCustomMessage, doLayout: %s", strategy) #layout using the specified strategy
    #the strategies that are able: cola, cose, circle, concentric, grid, breadthfirst, random, dagre, cose-bilkent
    doLayout(session, strategy)
  })

  observeEvent(input$hideSelection, ignoreInit = TRUE, {
    #all selected nodes and their edges are hidden
    hideSelection(session)
  })

    ...

  observeEvent(input$invertSelection, ignoreInit = TRUE, {
    #selected nodes and edges are hidden
    invertSelection(session)
  })

  observeEvent(input$showAll, ignoreInit = TRUE, {
    #all unselected and shown
    showAll(session)
  })

    ...
}
```

Listing B.14: Code for cyjShiny functions.

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Options of the Visualize your results tab ######
    ...
  #as it is desired that the hidden nodes in the visualize your results tab kept hidden in the next step
  #reactive expressions are defined
  hiddennodes <- reactiveVal() #for invert selected and remove selected buttons
  hiddennodes_builder <- reactiveVal()

  observeEvent(c(input$clearSelection, input$showAll),{
    if(identical(modality(),NULL)){
      #after a selection, the user can decide to unselect everything and so the hiddennodes reactive value is set to
           null
      hiddennodes(NULL)
    }else{
      hiddennodes_builder(NULL)
    }
  })

  #to keep the deletes in the next tab
  observeEvent(input$hideSelection, ignoreInit=TRUE, {
    getSelectedNodes(session) #when Remove Selected is pressed
  })

  observeEvent(input$selectedNodes, {
    if(input$hideSelection != 0){ #if Remove Selected has been pressed
      newNodes_hide <- input$selectedNodes;
```

```r
      if(identical(modality(),NULL)){ #templates
        if(is.null(hiddennodes())){ #considering non previous deletes
          df <- results_reactive() #the data frame of the graph
          for (node in newNodes_hide){
            if (node %in% df[,input$id_nodes]){ #nodes or edges separation
              df <- df[!df[,input$id_nodes]==node,] #keep those nodes non selected
            } else if (node %in% df[,input$id_edges]) { #here edges
              df <- df[!df[,input$id_edges]==node,]
            }
          }
        } else { #considering previous deletes
          df <- as.data.frame(hiddennodes())
          for (node in newNodes_hide){
            if (node %in% df[,input$id_nodes]){ #nodes or edges names separation
              df <- df[!df[,input$id_nodes]==node,]
            } else if (node %in% df[,input$id_edges]) {
              df <- df[!df[,input$id_edges]==node,]
            }
          }
        }
      } else { #query builder
        if(is.null(hiddennodes_builder())){
          df <- results_reactive_builder()
          for (node in newNodes_hide){
            if (node %in% df[,input$id_nodes]){ #nodes or edges names separation
              df <- df[!df[,input$id_nodes]==node,]
            } else if (node %in% df[,input$id_edges]) {
              df <- df[!df[,input$id_edges]==node,]
            }
          }
        } else {
          df <- as.data.frame(hiddennodes_builder())
          for (node in newNodes_hide){
            if (node %in% df[,input$id_nodes]){ #nodes or edges names separation
              df <- df[!df[,input$id_nodes]==node,]
            } else if (node %in% df[,input$id_edges]) {
              df <- df[!df[,input$id_edges]==node,]
            }
          }
        }
      }
      if (identical(modality(), NULL)){
        hiddennodes(df) #defining and saving the new data frame in a reactive value
      } else {
        hiddennodes_builder(df)
      }
      updateActionButton(session, "hideSelection", "Remove Selected") #update the action button
    } else {
      return()
    }
  })

  #for the template query pathway
  new_df <- eventReactive(c(input$hideSelection, input$clearSelection, input$showAll, input$goOverlaid1),{
    if (is.null(hiddennodes())){ #the user has not filtered the nodes
      df <- results_reactive()
    } else { #the user has done some filtering pressing hide selection action button
      df <- as.data.frame(hiddennodes())
    }
    df
  })

  #the same but for the built query pathway
  new_df_builder <- eventReactive(c(input$hideSelection, input$clearSelection, input$showAll, input$goOverlaid1),{
    if (is.null(hiddennodes_builder())){
      df <- results_reactive_builder()
    } else {
      df <- as.data.frame(hiddennodes_builder())
    }
    df
  })

    ...
}
```

Listing B.15: Code to keep deletes in the Overlay tab.

B.4.1.3. Wrapping cytoscape.js and downloading the network in png format.

```r
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ##### Wrapping cytoscape.js html widget in Visualize your results tab #####
  output$cyjShiny <- renderCyjShiny({
    if(identical(modality(),NULL)){
      g3 <- interaction_reactive()
      graph <- graphToJSON(g3)
      cyjShiny(graph, layoutName="cola", height = 600) #by default the cola layout is set
      #but doLayout() function is used to select other layouts
    }else{
      g3 <- interaction_reactive_builder()
      graph <- graphToJSON(g3)
      cyjShiny(graph, layoutName="cola", height = 600)
    }
  })
  #the html widget can be downloaded in png format
  observeEvent(input$downloadviewer, ignoreInit=TRUE, {
    file.name <- tempfile(fileext=".png") #.png
    savePNGtoFile(session, file.name)
  })
  observeEvent(input$pngData, ignoreInit=TRUE, {
    printf("received pngData")
    png.parsed <- fromJSON(input$pngData)
    substr(png.parsed, 1, 30) # [1] "data:image/png;base64,iVBORw0K"
    nchar(png.parsed)  # [1] 768714
    png.parsed.headless <- substr(png.parsed, 23, nchar(png.parsed))  # chop off the uri header
    png.parsed.binary <- base64decode(png.parsed.headless)
    if(nchar(input$filenameViewer)<1){
      printf(paste0("writing png to ","network",format(Sys.time(), "%m%d_%H%M"),".png"))
      conn <- file(paste0("network",format(Sys.time(), "%m%d_%H%M"),".png"), "wb")
    }else{
      printf(paste0("writing png to ",input$filenameViewer,".png"))
      conn <- file(paste0(input$filenameViewer,".png"), "wb")
    }
    writeBin(png.parsed.binary, conn)
    close(conn)
  })
  #info message of the download
  observeEvent(input$downloadviewer, {
    shinyalert(
      title = "The download is complete.",
      text = paste("You can find the file in ",getwd()," directory."),
      type = "info",
      closeOnEsc = FALSE,
      closeOnClickOutside = FALSE,
      html = FALSE,
      showCancelButton = FALSE,
      showConfirmButton = TRUE,
      inputType = "text",
      inputValue = "",
      inputPlaceholder = "",
      confirmButtonText = "OK",
      confirmButtonCol = "#AEDEF4",
      cancelButtonText = "Cancel",
      timer = 0,
      animation = TRUE,
      imageUrl = NULL,
      imageWidth = 100,
      imageHeight = 100,
      className = "", #modality reactive value is set to true
      callbackJS = NULL,
      inputId = "shinyalert"
    )
  })
    ...
}
```

Listing B.16: Code that displays the network and saves it as an image.

B.4.2. Overlay tab

B.4.2.1. From data frame results to edges and nodes dataframes

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Preparing nodes and edges dataframes for Overlay tab ######
  #the following function returns edges dataframe
  style_edges_reactive_func <- function(data_frame, id_nodes, id_edges){
    df <- data_frame
    nodes <- data.frame(id = unique(c(df[,id_nodes], df[,id_edges])), stringsAsFactors = FALSE) #only non-repited
    edges <- df %>%
      dplyr::select(source = all_of(id_nodes),
                    target = all_of(id_edges)) %>%
      dplyr::mutate(interaction = paste(source, '_', target))
  }

  style_edges_reactive <- eventReactive(c(input$t_choice,input$m.index_t1,input$m.index_t2, input$m.index_t3,
                                          input$m.index_t4, input$values_t1, input$values_t2, input$values_t3,
                                          input$values_t4, input$goResults,input$goOverlaid1, input$hideSelection,
                                          input$clearSelection, input$invertSelection, input$showAll,
                                          input$goInteraction, input$id_nodes,input$id_edges,input$nodes_attributes,
                                          input$edges_attributes),{

                                              #applying the previous function in the template query
                                              style_edges_reactive_func(new_df(),input$id_nodes, input$id_edges)

                                            })
  style_edges_reactive_builder <- eventReactive(c(input$b_choice,input$b_2_choice,input$b_3_choice,
                                                  input$b_4_choice,input$b_5_choice,
                                                  input$b_select_1,input$b_constraint_1,input$b_order,
                                                  input$desc_asc, input$b_select_2, input$b_constraint_2,
                                                  input$b_select_3, input$b_constraint_3, input$b_select_4,
                                                    input$b_constraint_4, input$b_select_5, input$b_constraint_5,
                                                    input$operator_0, input$value_0, input$operator_1, input$value_1,
                                                    input$operator_2, input$value_2, input$operator_3, input$value_3,
                                                    input$operator_4, input$value_4, input$operator_5, input$value_5,
                                                    input$goOverlaid1,input$hideSelection, input$clearSelection,
                                                    input$invertSelection, input$showAll, input$goBuilder,
                                                    input$goInteraction, input$id_nodes,input$id_edges,
                                                    input$nodes_attributes, input$edges_attributes),{

                                                      style_edges_reactive_func(new_df_builder(),input$id_nodes,input$id
                                                           _edges)

                                                    })
    ...
}
```

Listing B.17: Code that establishes the susceptible data to be filtered.

B.4.2.2. Options for the overlays and mapping.

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Options of the Overlay additional data tab ######
  observeEvent(input$goOverlaid1,{
    output$ui <- renderUI({
      if (is.null(input$mapping_question))
        return()

      # Depending on input$mapping_question, we'll generate a different
      # UI component and send it to the client.
      switch(input$mapping_question,
             "Yes, a colour gradient." = {
               box(width = 12,
                   colorSelectorInput("range_color1", "Choose the first colour:",
                                      choices = c("yellow",
                                                  "orange",
                                                  "red",
                                                  "magenta",
                                                  "blue",
                                                  "cyan",
                                                  "green")),
                   bsTooltip("range_color1", "This colour is the minimum."),
                   colorSelectorInput("range_color2", "Choose the second colour:",
                                      choices = rev(c("yellow",
                                                      "orange",
                                                      "red",
                                                      "magenta",
                                                      "blue",
                                                      "cyan",
                                                      "green"))),
                   bsTooltip("range_color2", "This colour is the maximum."),
                   sliderInput("range_color_numeric", "Specify the range of values to map:",min = 1, max = 1000,
                               value = c(200,500)),
                   bsTooltip("range_color_numeric", "Between the minimum and maximum value of the attribute.")
               )},
             "Yes, a size gradient." = {
               box(width = 12,
                   sliderInput("range_size", "Choose the extremes of the gradient:",
                               min = 10, max = 200, value = c(10,200), step = 5, round = TRUE),
                   bsTooltip("range_size", "These are the minimum and maximum sizes for the visual style."),
                   sliderInput("range_size_numeric", "Specify the range of values to map:",min = 1, max = 1000,
                               value = c(200,500)),
                   bsTooltip("range_size_numeric", "Between the minimum and maximum value of the attribute.")
               )},
             "No" = {return()}
      )
    })
  })
  observeEvent(c(input$mapping_question, input$gradient_id), {
    if(nchar(input$gradient_id)>2){ #if the answer to mapping question is yes (because no has two characters)
      updateSelectInput(session, "attr_color_grad",
                        choices = c(nodes_attr_reactive()))
      updateSelectInput(session, "attr_size_grad", #select node by attribute
                        choices = c(nodes_attr_reactive()))
    }
  })
  observeEvent(c(input$mapping_question, input$gradient_id), {
    if(nchar(input$gradient_id)>2){
      if(identical(modality(),NULL)){
        df <- new_df()
        ids = unique(c(df[,input$id_nodes], df[,input$id_edges]))
        data <- data.frame(id = character(),     # Create empty data frame
                           gradient_id = numeric(),
                           stringsAsFactors = FALSE)
        n=1
        for(i in ids){
          list <- df[df[, input$id_nodes] == i,]
          ii <- list[[input$gradient_id]]
          data[n,]<-list(i,as.numeric(ii[1]))
          n <- n+1
        }
      }else{
```

```r
        df <- as.data.frame(results_reactive_builder())
        ids = unique(c(df[,input$id_nodes], df[,input$id_edges]))
        data <- data.frame(id = character(),    # Create empty data frame
                           gradient_id = numeric(),
                           stringsAsFactors = FALSE)
        n=1
        for(i in ids){
          list <- df[df[, input$id_nodes] == i,]
          ii <- list[[input$gradient_id]]
          data[n,]<-list(i,as.numeric(ii[1]))
          n <- n+1
        }
      }
      #define a new scale considering the values of the data type
      updateSliderInput(session,"range_color_numeric",
                        min = min(data[,2],na.rm = TRUE), max = max(data[,2], na.rm = TRUE),
                        value = c(min(data[,2],na.rm = TRUE),max(data[,2], na.rm = TRUE))
      )
      updateSliderInput(session,"range_size_numeric",
                        min = min(data[,2], na.rm = TRUE), max = max(data[,2], na.rm = TRUE),
                        value = c(min(data[,2],na.rm = TRUE),max(data[,2], na.rm = TRUE))
      )
    }
  })

  observeEvent(input$goOverlaid1, {
    if(identical(modality(),NULL)){
      updateSelectInput(session, "selectName_3", #select node by attribute
                        choices = c("",c(nodes_attr_reactive())))
    }else{
      updateSelectInput(session, "selectName_3",
                        choices = c("",c(nodes_attr_reactive())))
    }

  })

  observeEvent(input$selectName_3, ignoreInit = TRUE, {
    if(identical(modality(),NULL)){
      df <- new_df()
      updateSelectInput(session, "selectName_3_attr", #select the attribute value
                        choices = c("",df[,input$selectName_3]))
    }else{
      df <- new_df_builder()
      updateSelectInput(session, "selectName_3_attr",
                        choices = c("",df[,input$selectName_3]))
    }
  })
    ...


  ###### Overlaying options ######
  observeEvent(c(input$hideSelection, input$clearSelection, input$showAll, input$goOverlaid1),{
    if(identical(modality(),NULL)){
      if(is.null(new_df())){
        return()
      } else {
        df <- new_df()
        options <- unique(c(df[,input$id_nodes], df[,input$id_edges]))
        #select nodes by ID
        updateSelectInput(session, "selectid",
                          choices = c("",options))
        updateSelectInput(session,"gradient_id",
                          choices = c("",c(nodes_attr_reactive())))
      }
    }else{
      if(is.null(new_df_builder())){
        return()
      } else {
        df <- new_df_builder()
        options <- unique(c(df[,input$id_nodes], df[,input$id_edges]))
        #select nodes by ID
        updateSelectInput(session, "selectid",
                          choices = c("",options))
        updateSelectInput(session,"gradient_id",
                          choices = c("",c(nodes_attr_reactive())))
      }
    }

  })

  observeEvent(input$select_parameter,{
    #options of the parameters that can be edited
    if(input$select_parameter=="shape"){
      options <- c("ellipse",
                   "triangle",
                   "round-triangle",
                   "rectangle",
                   "round-rectangle",
                   "bottom-round-rectangle",
                   "cut-rectangle",
                   "barrel",
                   "rhomboid",
                   "diamond",
                   "round-diamond",
                   "pentagon",
                   "round-pentagon",
                   "hexagon",
                   "round-hexagon",
                   "concave-hexagon",
                   "heptagon",
                   "round-heptagon",
                   "octagon",
                   "round-octagon",
                   "star",
                   "tag",
                   "round-tag",
                   "vee")
    }else if(input$select_parameter=="size"){
      options <- c("10","20","50","70","90","100","150") #size in pixels (unit)
    }else{
      options <- c("Orange"="#ff8c1a", #the option visible for the user to select is the name and the code is the
           argument
                   "Blue"="#99ccff",
                   "Dark Blue"="#0000cc",
                   "Forest Green"="#009900",
                   "Green"="#66ff66",
                   "Red"="#ff3300",
                   "Yellow"="#ffff4d",
                   "Purple"="#cc66ff",
                   "Pink"="#ff99cc",
                   "Grey"="#a6a6a6")

    }
    updateSelectInput(session, "select_parameter_option",
                      choices = c("", options))
  })
  ...
}
```

Listing B.18: Code that establishes the options.

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Styling nodes ######
  #function that modifies the nodes dataframe with the overlays
  style_nodes_reactive <- function(data_frame,data_frame_working, id_nodes, id_edges){
    df <- data_frame #original
    custom_df <- data_frame_working #overlays
    custom_df <- unique(custom_df[!is.null(custom_df[,"Parameter"]),])
    if( is.null(custom_df)){
      nodes <- data.frame(id = unique(c(df[,id_nodes], df[,id_edges]))) %>%
        dplyr::mutate(node_color = "#595959") %>% #background color by default
        dplyr::mutate(node_width = "10") %>% #size by defaults
        dplyr::mutate(node_height = "10") %>%
        dplyr::mutate(node_shape = "ellipse") #shape by default
    }else{
      nodes <- data.frame(id = unique(c(df[,id_nodes], df[,id_edges]))) %>%
        dplyr::mutate(node_color = "#595959") %>% #background color by default
        dplyr::mutate(node_width = "10") %>% #size by defaults
        dplyr::mutate(node_height = "10") %>%
        dplyr::mutate(node_shape = "ellipse") #shape by default

      custom_background <- subset(custom_df,Parameter == "background-color", select = c("Nodes","Selection")) #
           dataframe subset with only background-color
      custom_shape <- subset(custom_df,Parameter == "shape", select = c("Nodes","Selection")) #dataframe subset with
           only shape
      custom_size <- subset(custom_df, Parameter == "size", select = c("Nodes","Selection")) #dataframe subset with
           only size

      for (i in nodes$id) {
        if (i %in% c(custom_background$Nodes)){
          nodes$node_color[nodes$id == i] <- custom_background$Selection[custom_background$Nodes == i]
        }
        if (i %in% c(custom_size$Nodes)){
          nodes$node_width[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
          nodes$node_heigth[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
        }
        if (i %in% c(custom_shape$Nodes)){
          nodes$node_shape[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
        }
      }
      nodes
    }
  }
  style_nodes_reactive_mapping <- function(data_frame,data_frame_working, id_nodes, id_edges, attr_size_grad, attr_
       color_grad){
    if(input$mapping_question == "No"| nchar(input$gradient_id)<2){
      df <- data_frame #original
      custom_df <- data_frame_working #overlays
      custom_df <- unique(custom_df[!is.null(custom_df[,"Parameter"]),])
      if( is.null(custom_df)){
        nodes <- data.frame(id = unique(c(df[,id_nodes], df[,id_edges]))) %>%
          dplyr::mutate(node_color = "#595959") %>% #background color by default
          dplyr::mutate(node_width = "10") %>% #size by defaults
          dplyr::mutate(node_height = "10") %>%
          dplyr::mutate(node_shape = "ellipse") #shape by default
      }else{
        nodes <- data.frame(id = unique(c(df[,id_nodes], df[,id_edges]))) %>%
          dplyr::mutate(node_color = "#595959") %>% #background color by default
          dplyr::mutate(node_width = "10") %>% #size by defaults
          dplyr::mutate(node_height = "10") %>%
          dplyr::mutate(node_shape = "ellipse") #shape by default

        custom_background <- subset(custom_df,Parameter == "background-color", select = c("Nodes","Selection")) #
             dataframe subset with only background-color
```

```r
        custom_shape <- subset(custom_df,Parameter == "shape", select = c("Nodes","Selection")) #dataframe subset with only shape
        custom_size <- subset(custom_df, Parameter == "size", select = c("Nodes","Selection")) #dataframe subset with only size

        for (i in nodes$id) {
          if (i %in% c(custom_background$Nodes)){
            nodes$node_color[nodes$id == i] <- custom_background$Selection[custom_background$Nodes == i]
          }
          if (i %in% c(custom_size$Nodes)){
            nodes$node_width[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
            nodes$node_heigth[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
          }
          if (i %in% c(custom_shape$Nodes)){
            nodes$node_shape[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
          }
        }
        nodes
      }
    }else{
      df <- data_frame
      if(input$mapping_question == "Yes, a colour gradient."){
        ids = unique(c(df[,id_nodes], df[,id_edges]))
        data <- data.frame(id = character(),    # Create empty data frame
                           gradient_id = numeric(),
                           stringsAsFactors = FALSE)
        n=1
        for(i in ids){
          list <- df[df[, id_nodes] == i,]
          ii <- list[[attr_color_grad]]
          data[n,]<-list(i,as.numeric(ii[1]))
          n <- n+1
        }
      }else{
        ids = unique(c(df[,id_nodes], df[,id_edges]))
        data <- data.frame(id = character(),    # Create empty data frame
                           gradient_id = numeric(),
                           stringsAsFactors = FALSE)
        n=1
        for(i in ids){
          list <- df[df[, id_nodes] == i,]
          ii <- list[[attr_size_grad]]
          data[n,]<-list(i,as.numeric(ii[1]))
          n <- n+1
        }
      }
      custom_df <- data_frame_working #overlays
      custom_df <- unique(custom_df[!is.null(custom_df[,"Parameter"]),])
      if( is.null(custom_df)){
        nodes <- data.frame(id = unique(c(df[,id_nodes], df[,id_edges])), gradient = data[,2]) %>%
          dplyr::mutate(node_color = "#595959") %>% #background color by default
          dplyr::mutate(node_width = "10") %>% #size by defaults
          dplyr::mutate(node_height = "10") %>%
          dplyr::mutate(node_shape = "ellipse") #shape by default
      }else{
        nodes <- data.frame(id = unique(c(df[,id_nodes], df[,id_edges])), gradient = data[,2]) %>%
          dplyr::mutate(node_color = "#595959") %>% #background color by default
          dplyr::mutate(node_width = "10") %>% #size by defaults
          dplyr::mutate(node_height = "10") %>%
          dplyr::mutate(node_shape = "ellipse") #shape by default

        custom_background <- subset(custom_df,Parameter == "background-color", select = c("Nodes","Selection")) #dataframe subset with only background-color
        custom_shape <- subset(custom_df,Parameter == "shape", select = c("Nodes","Selection")) #dataframe subset with only shape
        custom_size <- subset(custom_df, Parameter == "size", select = c("Nodes","Selection")) #dataframe subset with only size

        for (i in nodes$id) {
          if (i %in% c(custom_background$Nodes)){
            nodes$node_color[nodes$id == i] <- custom_background$Selection[custom_background$Nodes == i]
          }
          if (i %in% c(custom_size$Nodes)){
            nodes$node_width[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
            nodes$node_heigth[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
          }
          if (i %in% c(custom_shape$Nodes)){
            nodes$node_shape[nodes$id == i] <- custom_shape$Selection[custom_shape$Nodes == i]
          }
        }
        nodes
      }
    }
  }

  style_custom_nodes_reactive_gradient <- eventReactive(c( ... ),{
        style_nodes_reactive_mapping(new_df(),values$dfWorking, input$id_nodes, input$id_edges, input$gradient_id, input$gradient_id)
                                                          })

  style_custom_nodes_reactive_builder <- eventReactive(c( ... ),{
        style_nodes_reactive_mapping(new_df_builder(), values_builder$dfWorking_builder, input$id_nodes, input$id_edges, input$gradient_id, input$gradient_id)
                                                         })
  ...
}
```

Listing B.19: Code that modifies the nodes data frame with the overlays.

B.4.2.3. Cytoscape and node_style

```r
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Cytoscape network chart ######
  observeEvent(c(input$hideSelection, input$clearSelection, input$showAll, input$goOverlaid1),{
    if(identical(modality(),NULL)){
      #cytoscape network chart
      #nodes and edges data frames are the arguments of the function
      plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
        cytoscape::layout('breadthfirst', directed = TRUE) %>% #by default, once initialized
        panzoom()
      saveWidget(plotInput, "temp.html", selfcontained = FALSE)
      output$network <- renderCytoscape({
        # draw the network
        cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
          cytoscape::layout('breadthfirst', directed = TRUE) %>%
          panzoom()

      })

    }else{
      plotInput <- cytoscape(nodes = style_custom_nodes_reactive_builder(), edges = style_edges_reactive_builder())
           %>%
        cytoscape::layout('breadthfirst', directed = TRUE) %>%
        panzoom()
      saveWidget(plotInput, "temp.html", selfcontained = FALSE)
      output$network <- renderCytoscape({
        # draw the network
        cytoscape(nodes = style_custom_nodes_reactive_builder(), edges = style_edges_reactive_builder()) %>%
          cytoscape::layout('breadthfirst', directed = TRUE) %>%
          panzoom()

      })
    }
  })
    ...
}
```

Listing B.20: Code for the Cytoscape Network when the user first visualize the Overlay tab.

```r
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  #for templates
  rv <- NULL
  values <- reactiveValues(dfWorking = rv) #a dataframe where the edited parameters are saved in rows, initialized
       here
  #same for builder
  rv_builder <- NULL
  rv_select <- NULL
  values_builder <- reactiveValues(dfWorking_builder = rv_builder)
  values_select <- reactiveValues(dfSelect = rv_select)
    ...
  ###### Applying the overlays ######
  observeEvent(c(input$button_set,input$layoutcytoscape, input$downloadstyle),{
    if(identical(modality(),NULL)){
      df <- new_df()
      node<-NULL
      node_i<-NULL
      try(node_i<-df[,input$selectName_3]==input$selectName_3_attr, silent = TRUE)
      try(node<-df[node_i,input$id_nodes], silent = TRUE) #getting the ID
      #First the dfWorking data frame is written with the user options
      if(is.null(input$selectid)){ #the user has not decided to edit edit nodes by ID
        if(is.null(input$selectName_3_attr)){ #if the user has not decided to edit any parameter
          strategy <- input$layoutcytoscape #the user only has changed the layout strategy
        }else{ #the user has decided to edit nodes by attribute
          for (element in data.frame(rbind(node,input$selectName_3_attr))){
            #the dfworking dataframe is constructed defining 4 columns: Nodes, Attribute, Parameter, Selection
            df_2 <- data.frame("Nodes"=element[1], "Attribute"=paste0(input$selectName_3," = ", element[2]), "
                 Parameter"=input$select_parameter,"Selection"=input$select_parameter_option)
            values$dfWorking <- rbind(values$dfWorking, df_2)
            df_2 <- NULL
          }
          #the following piece of code resets the attributes
          isolate({
            updateSelectInput(session, "selectName_3",
                              choices = c("",c(nodes_attr_reactive())))

          })
        }
      }else{ #the user has decided to edit nodes by ID
        for (element in input$selectid){
          df_2 <- data.frame("Nodes"=element,"Attribute"="ID", "Parameter"=input$select_parameter,"Selection"=input$
               select_parameter_option)
          values$dfWorking <- rbind(values$dfWorking, df_2)
          df_2 <- NULL
        }
      }
      #the following piece of code resets the IDs and so the user can create a new overlay
      isolate({ #to read reactive values without establishing a relationship with the caller (non re-execution)
        options <- unique(c(df[,input$id_nodes], df[,input$id_edges]))
        updateSelectInput(session, "selectid",
                          choices = c("",options))
      })
      #the following piece of code resets the editable parameters
      isolate({
        updateSelectInput(session,"select_parameter",
                          choices = c("", "Background colour"="background-color",
                                      "Shape"="shape",
                                      "Size"="size"
                          ))
      })
      #the following piece of code resets the attributes
      isolate({
        updateSelectInput(session, "selectName_3",
                          choices = c("",c(nodes_attr_reactive())))

      })
      #code to call cytoscape function and update the network
      if(is.null(values$dfWorking)){ #if there are no overlays
        if(input$mapping_question == "No"  | nchar(input$gradient_id)<2){
          strategy <- input$layoutcytoscape
          printf("about to sendCustomMessage, layout: %s", strategy) #only a new layout is defined by the user
          if(strategy=="cola"){
            plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
              cola_layout(avoidOverlap = TRUE) %>% #special function when the layout is "cola"
              panzoom()
            saveWidget(plotInput, "temp.html", selfcontained = FALSE)
            output$network <- renderCytoscape({
              cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                cola_layout(avoidOverlap = TRUE) %>%
                panzoom()
            })
          }else{
            plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
              cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
              panzoom()
            saveWidget(plotInput, "temp.html", selfcontained = FALSE)
            output$network <- renderCytoscape({
              cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
                panzoom()
            })
          }
        }else{
          if(input$mapping_question == "Yes, a colour gradient."){ #the user wants to apply a colour gradient
            if(strategy == "cola"){
              plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                #the first argument, previous = sign, of node_style is a recognised cytoscape node style name
                #the second argument is the label given in the dataframe
                node_style('background-color' = paste0('mapData(gradient,',
                                                       as.character(input$range_color_numeric[1]),', ',
                                                       as.character(input$range_color_numeric[2]),', ',
                                                       input$range_color1,', ',
                                                       input$range_color2,')')) %>%

                cola_layout(avoidOverlap = TRUE) %>%
                panzoom()
            }else{
              plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                node_style('background-color' = paste0('mapData(gradient,',
                                                       as.character(input$range_color_numeric[1]),',',
                                                       as.character(input$range_color_numeric[2]),',',
                                                       input$range_color1,',',
                                                       input$range_color2,')')) %>%

                cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
                panzoom()
            }
            saveWidget(plotInput, "temp.html", selfcontained = FALSE)
            output$network <- renderCytoscape({
              strategy <- input$layoutcytoscape
              printf("about to sendCustomMessage, layout: %s", strategy)
              if(strategy == "cola"){
                cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                  node_style('background-color' = paste0('mapData(gradient,',
                                                         as.character(input$range_color_numeric[1]),',',
                                                         as.character(input$range_color_numeric[2]),',',
                                                         input$range_color1,',',
                                                         input$range_color2,')')) %>%

                  cola_layout(avoidOverlap = TRUE) %>%
                  panzoom()
              }else{
                cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                  node_style('background-color' = paste0('mapData(gradient,',
                                                         as.character(input$range_color_numeric[1]),',',
                                                         as.character(input$range_color_numeric[2]),',',
                                                         input$range_color1,',',
                                                         input$range_color2,')')) %>%

                  cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
                  panzoom()
              }
            })
          }else{ #the user wants to apply a size gradient
            if(strategy == "cola"){
              plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                #the first argument, previous = sign, of node_style is a recognised cytoscape node style name
                #the second argument is the label given in the dataframe

                node_style('width' = paste0('mapData(gradient,',
                                            as.character(input$range_size_numeric[1]),',',
                                            as.character(input$range_size_numeric[2]),',',
                                            input$range_size[1],',',
                                            input$range_size[2],')')) %>%
                node_style('height' = paste0('mapData(gradient,',
                                             as.character(input$range_size_numeric[1]),',',
                                             as.character(input$range_size_numeric[2]),',',
                                             input$range_size[1],',',
                                             input$range_size[2],')')) %>%

                cola_layout(avoidOverlap = TRUE) %>%
                panzoom()
            }else{
              plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive())
                   %>%

                node_style('width' = paste0('mapData(gradient,',
                                            as.character(input$range_size_numeric[1]),',',
                                            as.character(input$range_size_numeric[2]),',',
                                            input$range_size[1],',',
                                            input$range_size[2],')')) %>%
                node_style('height' = paste0('mapData(gradient,',
                                             as.character(input$range_size_numeric[1]),',',
                                             as.character(input$range_size_numeric[2]),',',
                                             input$range_size[1],',',
                                             input$range_size[2],')')) %>%

                cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
                panzoom()
            }
            saveWidget(plotInput, "temp.html", selfcontained = FALSE)
            output$network <- renderCytoscape({
              strategy <- input$layoutcytoscape
              printf("about to sendCustomMessage, layout: %s", strategy)
              if(strategy == "cola"){
                cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%

                  node_style('width' = paste0('mapData(gradient,',
                                              as.character(input$range_size_numeric[1]),',',
                                              as.character(input$range_size_numeric[2]),',',
                                              input$range_size[1],',',
                                              input$range_size[2],')')) %>%
                  node_style('height' = paste0('mapData(gradient,',
                                               as.character(input$range_size_numeric[1]),',',
                                               as.character(input$range_size_numeric[2]),',',
                                               input$range_size[1],',',
                                               input$range_size[2],')')) %>%

                  cola_layout(avoidOverlap = TRUE) %>%
                  panzoom()
              }else{
                cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%

                  node_style('width' = paste0('mapData(gradient,',
                                              as.character(input$range_size_numeric[1]),',',
                                              as.character(input$range_size_numeric[2]),',',
                                              input$range_size[1],',',
                                              input$range_size[2],')')) %>%
                  node_style('height' = paste0('mapData(gradient,',
                                               as.character(input$range_size_numeric[1]),',',
                                               as.character(input$range_size_numeric[2]),',',
                                               input$range_size[1],',',
                                               input$range_size[2],')')) %>%

                  cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
                  panzoom()
              }
            })
          }
        }
      }else{ #with overlays
        strategy <- input$layoutcytoscape
        if(input$mapping_question == "No"| nchar(input$gradient_id)<2){
          if(strategy == "cola"){
            plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
              #the first argument, previous = sign, of node_style is a recognised cytoscape node style name
              #the second argument is the label given in the dataframe
              node_style('background-color' = 'data(node_color)') %>%
              node_style('shape' = 'data(node_shape)') %>%
              node_style('width' = 'data(node_width)') %>% #size defines width and heigth
              node_style('height' = 'data(node_height)') %>%
              cola_layout(avoidOverlap = TRUE) %>%
              panzoom()
          }else{
            plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
              node_style('background-color' = 'data(node_color)') %>%
              node_style('shape' = 'data(node_shape)') %>%
              node_style('width' = 'data(node_width)') %>%
              node_style('height' = 'data(node_height)') %>%
              cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
              panzoom()
          }
          saveWidget(plotInput, "temp.html", selfcontained = FALSE)
          output$network <- renderCytoscape({
            strategy <- input$layoutcytoscape
```

```
    printf("about to sendCustomMessage, layout: %s", strategy)
    if(strategy == "cola"){
      cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
        node_style('background-color' = 'data(node_color)') %>%
        node_style('shape' = 'data(node_shape)') %>%
        node_style('width' = 'data(node_width)') %>%
        node_style('height' = 'data(node_height)') %>%
        cola_layout(avoidOverlap = TRUE) %>%
        panzoom()
    }else{
      cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
        node_style('background-color' = 'data(node_color)') %>%
        node_style('shape' = 'data(node_shape)') %>%
        node_style('width' = 'data(node_width)') %>%
        node_style('height' = 'data(node_height)') %>%
        cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
        panzoom()
    }
  })
}else{
  if(input$mapping_question == "Yes, a colour gradient."){
    if(strategy == "cola"){
      plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
        #the first argument, previous = sign, of node_style is a recognised cytoscape node style name
        #the second argument is the label given in the dataframe
        node_style('background-color' = paste0('mapData(gradient,',
                                               as.character(input$range_color_numeric[1]),',',
                                               as.character(input$range_color_numeric[2]),',',
                                               input$range_color1,',',
                                               input$range_color2,')')) %>%
        node_style('shape' = 'data(node_shape)') %>%
        node_style('width' = 'data(node_width)') %>%
        node_style('height' = 'data(node_height)') %>%
        cola_layout(avoidOverlap = TRUE) %>%
        panzoom()
    }else{
      plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
        node_style('background-color' = paste0('mapData(gradient,',
                                               as.character(input$range_color_numeric[1]),',',
                                               as.character(input$range_color_numeric[2]),',',
                                               input$range_color1,',',
                                               input$range_color2,')')) %>%
        node_style('shape' = 'data(node_shape)') %>%
        node_style('width' = 'data(node_width)') %>%
        node_style('height' = 'data(node_height)') %>%
        cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
        panzoom()
    }
    saveWidget(plotInput, "temp.html", selfcontained = FALSE)
    output$network <- renderCytoscape({
      strategy <- input$layoutcytoscape
      printf("about to sendCustomMessage, layout: %s", strategy)
      if(strategy == "cola"){
        cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
          node_style('background-color' = paste0('mapData(gradient,',
                                                 as.character(input$range_color_numeric[1]),',',
                                                 as.character(input$range_color_numeric[2]),',',
                                                 input$range_color1,',',
                                                 input$range_color2,')')) %>%
          node_style('shape' = 'data(node_shape)') %>%
          node_style('width' = 'data(node_width)') %>%
          node_style('height' = 'data(node_height)') %>%
          cola_layout(avoidOverlap = TRUE) %>%
          panzoom()
      }else{
        cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
          node_style('background-color' = paste0('mapData(gradient,',
                                                 as.character(input$range_color_numeric[1]),',',
                                                 as.character(input$range_color_numeric[2]),',',
                                                 input$range_color1,',',
                                                 input$range_color2,')')) %>%
          node_style('shape' = 'data(node_shape)') %>%
          node_style('width' = 'data(node_width)') %>%
          node_style('height' = 'data(node_height)') %>%
          cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
          panzoom()
      }
    })
  }else{
    if(strategy == "cola"){
      plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
        #the first argument, previous = sign, of node_style is a recognised cytoscape node style name
        #the second argument is the label given in the dataframe
        node_style('background-color' = 'data(node_color)') %>%
        node_style('shape' = 'data(node_shape)') %>%
        node_style('width' = paste0('mapData(gradient,',
                                    as.character(input$range_size_numeric[1]),',',
                                    as.character(input$range_size_numeric[2]),',',
                                    input$range_size[1],',',
                                    input$range_size[2],')')) %>%
        node_style('height' = paste0('mapData(gradient,',
                                     as.character(input$range_size_numeric[1]),',',
                                     as.character(input$range_size_numeric[2]),',',
                                     input$range_size[1],',',
                                     input$range_size[2],')')) %>%
        cola_layout(avoidOverlap = TRUE) %>%
                panzoom()
            }else{
              plotInput <- cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                node_style('background-color' = 'data(node_color)') %>%
                node_style('shape' = 'data(node_shape)') %>%
                node_style('width' = paste0('mapData(gradient,',
                                            as.character(input$range_size_numeric[1]),',',
                                            as.character(input$range_size_numeric[2]),',',
                                            input$range_size[1],',',
                                            input$range_size[2],')')) %>%
                node_style('height' = paste0('mapData(gradient,',
                                             as.character(input$range_size_numeric[1]),',',
                                             as.character(input$range_size_numeric[2]),',',
                                             input$range_size[1],',',
                                             input$range_size[2],')')) %>%
                cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
                panzoom()
            }
            saveWidget(plotInput, "temp.html", selfcontained = FALSE)
            output$network <- renderCytoscape({
              strategy <- input$layoutcytoscape
              printf("about to sendCustomMessage, layout: %s", strategy)
              if(strategy == "cola"){
                cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                  node_style('background-color' = 'data(node_color)') %>%
                  node_style('shape' = 'data(node_shape)') %>%
                  node_style('width' = paste0('mapData(gradient,',
                                              as.character(input$range_size_numeric[1]),',',
                                              as.character(input$range_size_numeric[2]),',',
                                              input$range_size[1],',',
                                              input$range_size[2],')')) %>%
                  node_style('height' = paste0('mapData(gradient,',
                                               as.character(input$range_size_numeric[1]),',',
                                               as.character(input$range_size_numeric[2]),',',
                                               input$range_size[1],',',
                                               input$range_size[2],')')) %>%
                  cola_layout(avoidOverlap = TRUE) %>%
                  panzoom()
              }else{
                cytoscape(nodes = style_custom_nodes_reactive_gradient(), edges = style_edges_reactive()) %>%
                  node_style('background-color' = 'data(node_color)') %>%
                  node_style('shape' = 'data(node_shape)') %>%
                  node_style('width' = paste0('mapData(gradient,',
                                              as.character(input$range_size_numeric[1]),',',
                                              as.character(input$range_size_numeric[2]),',',
                                              input$range_size[1],',',
                                              input$range_size[2],')')) %>%
                  node_style('height' = paste0('mapData(gradient,',
                                               as.character(input$range_size_numeric[1]),',',
                                               as.character(input$range_size_numeric[2]),',',
                                               input$range_size[1],',',
                                               input$range_size[2],')')) %>%
                  cytoscape::layout(strategy, avoidOverlap = TRUE) %>%
                  panzoom()
              }
            })
          }
        }
        node <- NULL
        node_i <- NULL
      }}else{ #exactly the same but with the data from a built query
        ...
      }
  })
    ...
}
```

Listing B.21: Code to apply the overlays to the network.

B.4.2.4. Saving the results: image and zip folder.

```
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Saving the results from overlaying ######
  observe({ #saving an image in png format
    output$ggsave_graph <- downloadHandler(
      filename = function() {ifelse(input$imagenameStyle=="", paste0("network",format(Sys.time(), "%m%d_%H%M"),".png"), #png
                                    paste0(input$imagenameStyle,".png"))},
      content = function(file) {
        webshot("temp.html", file = file , cliprect = "viewport") #screenshot of the temporal html file saved with the function saveWidget()
        #as I am using this function I cannot save in vectorial format (EPS)
      }
    )
  })

  observe({#saving a zip folder with the results (csv of the previous step), customization (csv), ids (csv), network (json)
    if(identical(modality(),NULL)){
      output$downloadstyle <- downloadHandler(
```

```r
        filename <- function() { #filename of the zip and extension
          ifelse(input$filenameStyle=="",paste0("workflow_final",format(Sys.time(), "%m%d_%H%M"),".zip"), # default name of the zip
                 paste0(input$filenameStyle,".zip"))
        },
        content <- function(file) {
          temp <- tempdir() # Set a temp dir
          setwd(tempdir()) #the content is written to the temp dir
          # Create the files
          if(is.null(values$dfWorking)){
            results_table_path <- paste0("results_",format(Sys.time(), "%m%d_%H%M"),".csv") #month day _ hour minute
            customization_table_path <- paste0("customization_",format(Sys.time(), "%m%d_%H%M"),".txt")
            ids_path <- paste0("ids_",format(Sys.time(), "%m%d_%H%M"),".csv")
            json_path <- paste0("network_",format(Sys.time(), "%m%d_%H%M"),".json")

            utils::write.csv(new_df(), results_table_path, row.names = TRUE)
            write("", customization_table_path)
            utils::write.csv(data.frame(nodes=input$id_nodes, edges=input$id_edges), ids_path)
            write(dataFramesToJSON(style_edges_reactive(), style_custom_nodes_reactive_gradient()),json_path)
            # Create a zip of the data
            zip::zipr(zipfile = file, files = c(results_table_path,
                                                customization_table_path,
                                                ids_path, json_path))
          }else{  #overlaying by the user
            results_table_path <- paste0("results_",format(Sys.time(), "%m%d_%H%M"),".csv")
            customization_table_path <- paste0("customization_",format(Sys.time(), "%m%d_%H%M"),".csv")
            ids_path <- paste0("ids_",format(Sys.time(), "%m%d_%H%M"),".csv")
            json_path <- paste0("network_",format(Sys.time(), "%m%d_%H%M"),".json")

            utils::write.csv(new_df(), results_table_path, row.names = TRUE)
            dt <- values$dfWorking
            dt <- unique(dt[!is.null(dt[,"Parameter"]),])
            utils::write.csv(dt, customization_table_path, row.names = TRUE)
            utils::write.csv(data.frame(nodes=input$id_nodes, edges=input$id_edges), ids_path)
            write(dataFramesToJSON(style_edges_reactive(), style_custom_nodes_reactive_gradient()),json_path)
            # Create a zip of the data
            zip::zipr(zipfile = file, files = c(results_table_path, customization_table_path, ids_path, json_path))
          }
        },
        contentType = "application/zip")
    }else{ #the same but for the builder path
      output$downloadstyle <- downloadHandler(
        filename <- function() {
          ifelse(input$filenameStyle=="",paste0("workflow_final",format(Sys.time(), "%m%d_%H%M"),".zip"), # default name of the zip
                 paste0(input$filenameStyle,".zip"))
        },
        content <- function(file) {
          temp <- tempdir() # Set a temp dir
          setwd(tempdir())
          # Create the files
          if(is.null(values_builder$dfWorking_builder)){
            results_table_path <- paste0("results_",format(Sys.time(), "%m%d_%H%M"),".csv")
            customization_table_path <- paste0("customization_",format(Sys.time(), "%m%d_%H%M"),".txt")
            ids_path <- paste0("ids_",format(Sys.time(), "%m%d_%H%M"),".csv")
            json_path <- paste0("network_",format(Sys.time(), "%m%d_%H%M"),".json")

            utils::write.csv(new_df_builder(), results_table_path, row.names = TRUE)
            write("", customization_table_path)
            utils::write.csv(data.frame(nodes=input$id_nodes, edges=input$id_edges), ids_path)
            write(dataFramesToJSON(style_edges_reactive_builder(), style_custom_nodes_reactive_builder()),json_path)
            # Create a zip of the data
            zip::zipr(zipfile = file, files = c(results_table_path,
                                                customization_table_path,
                                                ids_path, json_path))
          }else{
            results_table_path <- paste0("results_",format(Sys.time(), "%m%d_%H%M"),".csv")
            customization_table_path <- paste0("customization_",format(Sys.time(), "%m%d_%H%M"),".csv")
            ids_path <- paste0("ids_",format(Sys.time(), "%m%d_%H%M"),".csv")
            json_path <- paste0("network_",format(Sys.time(), "%m%d_%H%M"),".json")

            utils::write.csv(new_df_builder(), results_table_path, row.names = TRUE)
            dt <- values_builder$dfWorking_builder
            dt <- unique(dt[!is.null(dt[,"Parameter"]),])
            utils::write.csv(dt, customization_table_path, row.names = TRUE)
            utils::write.csv(data.frame(nodes=input$id_nodes, edges=input$id_edges), ids_path)
            write(dataFramesToJSON(style_edges_reactive_builder(), style_custom_nodes_reactive_builder()),json_path)
            # Create a zip of the data
            zip::zipr(zipfile = file, files = c(results_table_path, customization_table_path, ids_path, json_path))
          }
        },
        contentType = "application/zip")
    }
  })

    ...
}
```

Listing B.22: Code to save the results from the Overlay tab.

B.4.3. Saved Networks tab

B.4.3.1. Unzip and display saved networks.

```r
#Server: where the data is processed-----
server <- function(input, output, session){
    ...
  ###### Unzip and display saved Networks ######
  observeEvent(input$unzip,{
    filename_glob <- "*0"
    output_dir = tempdir() #dir where saves unzip data
    setwd(tempdir())
    # Unzip data in output_dir
    unzip <- utils::unzip(input$file$datapath, list = TRUE, overwrite = TRUE, exdir = output_dir)
    ls_content <- unzip$Name
    print(ls_content)
    # Displaying the results from saved queries in a data table
    output$resultstable_save <- renderDT({
      results <- datatable(as.data.frame(read.csv(ls_content[1])), fillContainer = TRUE, rownames = FALSE, options = list(
        pageLength = 25, autoWidth = TRUE))
      results
    })
    if(str_sub(ls_content[2],-1)=="t"){ #if the format from the second file is .txt (last chr is t) the network has not got overlays
      df <- NULL
    }else{
      df <- read.csv(ls_content[2])
    }
    output$network_saved <- renderCytoscape({
      cytoscape(nodes = style_nodes_reactive(read.csv(ls_content[1]),df, read.csv(ls_content[3])$nodes, read.csv(ls_content[3])$edges),
              edges = style_edges_reactive_func(read.csv(ls_content[1]),read.csv(ls_content[3])$nodes, read.csv(ls_content[3])$edges)) %>%
        node_style('background-color' = 'data(node_color)') %>%
        node_style('shape' = 'data(node_shape)') %>%
        node_style('width' = 'data(node_width)') %>%
        node_style('height' = 'data(node_height)') %>%
        cola_layout(avoidOverlap = TRUE) %>% #by default cola layout
        panzoom()
    })
  })
    ...
}
```

Listing B.23: Code to unzip and display saved networks.

B.5. User guide

InterMineR Cytoscape Interface

InterMine
University of Cambridge

Celia Sánchez Laorden
csanchla8@alumnes.ub.edu
csl.celiasanchez@gmail.com

Source: InterMineR Cytoscape Interface is on Github

April, 2021

This interface wants to be a guide to run queries and interpret them with the intuitive Cytoscape visualizations without prior software experience. It facilitates understanding and communication of relevant relationships between different biological Data Classes.

Contents

1 Requirements

InterMineR Cytoscape is an interface created with Shiny. All code is available from: Github. To run the Shiny app, first, make sure that you have installed:

1. R Studio version 4.0.3 or above.
2. All the files and the `"www"` folder from the GitHub repository. Unzip the folder. Set your working directory to the unzipped folder once in R Studio (see setwd function).
3. All of the packages from the `Packages.R` file in R Studio. In order to install them correctly, run the installation for each package in turn.

Open R Studio, open the file `app.R` and open the file `workspace_app.RData`. Then, press `"RunApp"`.

2 Capabilities

Using this app you will be able to:

1. Run queries using any template from all registered InterMine instances in one place.
2. Advanced users can use a flexible query interface to construct their own data mining queries.
3. Display and export the results in a table.
4. A set of network visualization tools from Cytoscape domains enrich the interpretation of the results.
5. Further options allow customization of the Cytoscape Networks.
6. Store your visualizations in JSON format and display saved Networks.

3 Basic Usage

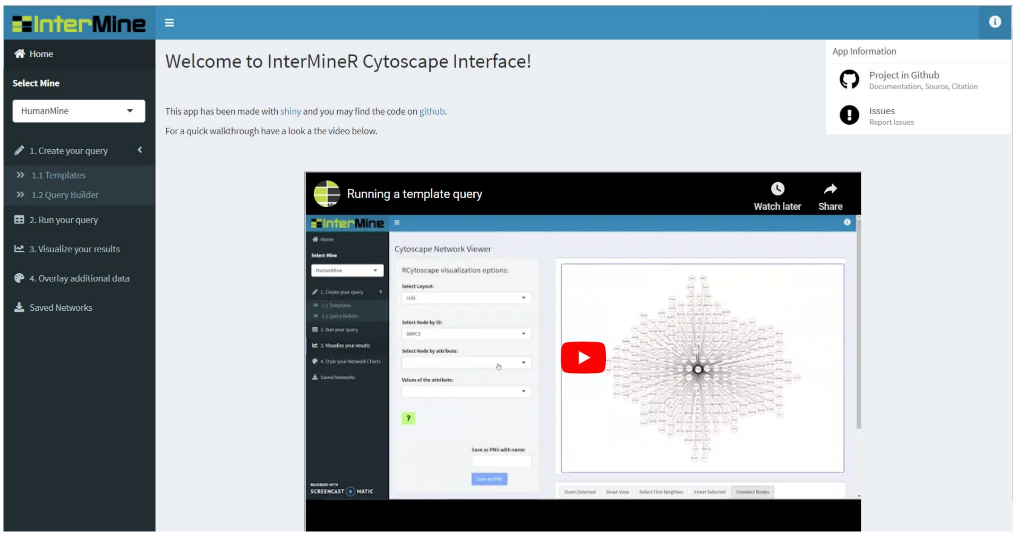
The InterMineR Cytoscape Interface divides the tasks into seven tabs:

- **Home**: contains a short walk-trough the app.
- **Create your query**: here you can select either **Templates**, which allows you to select predefined queries. Or **Query Builder**, which provides a tool to flexibly create your own queries.
- **Run your query**: displays the results from the query previously created.
- **Visualize your results**: is a Cytoscape Network viewer.
- **Overlay additional data**: is the tool to style the Network chart.
- **Saved Networks**: enables you to visualize the results you have saved from past queries.

3.1 Overview

On the left side of the screen there is a sidebar menu, where in addition to the tab menu, a select list enables you to choose a registered InterMine instance. In the upper right corner, there is an information button that takes you to the source project in GitHub and redirects you to a place where you can expose any issue you find.

In each tab panel, you will find a help button. When you click to the question mark **?** icon step-by-step instructions will be displayed. Some hints are also visible by hovering your mouse over the buttons.



Welcome to InterMineR Cytoscape Interface!

This app has been made with shiny and you may find the code on github.

For a quick walkthrough have a look a the video below.

Figure 1: InterMineR Cytoscape Interface Home tab.

Each tab will be explained in more detail below:

3.2 1.1 Templates tab

You first will need to select a query from the list of templates. At this point, you will see the predefined constraints of the template in the main panel of the tab. You can change the value by default and this way modify the constraints.

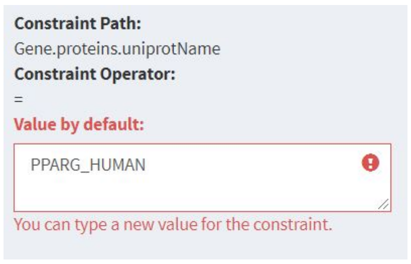

Constraint Path:
Gene.proteins.uniprotName
Constraint Operator:
=
Value by default:
PPARG_HUMAN
You can type a new value for the constraint.

Figure 2: Pre-defined constraint for the Template `Protein-->Interactions` from HumanMine.

A summary table of the constraints selected is displayed. Each time you select a new template query, all the modifications are deleted, and you can start again. To go to the next section, click `Go to Results` at the lower right corner.

3.3 1.2 Query Builder tab

The Query Builder modality is only encouraged for experienced users. To start building a query from scratch, first, you will need to define a Data Class. Then, you must set which attributes you want to see in the results. This also defines the type of sorting which will be used to order the retrieved data.frame. At the bottom of the page, you can change the predefined choice of ordering and select if you want to sort the results in ascending or descending order. Setting a constraint for the first Data Class is optional, but take into account that at some point you will need to define one. At this point, if you are defining a constraint you can type multiple values separated by commas. Just a clarification: the constraint operator and value(s) that you enter in the boxes below your first Data Class choice run against this class. If you press the `Constraints` button you can set a constraint against an attribute from the first Data Class. You can only select one attribute, otherwise you will get an error.

Press the `Set` button if you want to add a second level, dependent on your first Data Class choice, and overlay extra data. In the consecutive steps, one checkbox tree is displayed for the data type to be returned and another one for data types to set constraints. In the first tree, `Type of data to be returned`, you can select the data you want to see in the results table. If you want to set a third level, pressing the `Set` button below the trees, you could need to select the data types in the second level that you do not want to see displayed in the results table. Do not worry, you can delete these data types after setting the third and consecutive levels pressing `Set Query` button. When you press this button, a emergent window will appear with all the values that are going to be seen in the results table. By selecting the ones you want to remove and pressing `Delete Rows` button you will get rid off any undesired data type.

Once you have achieved the 3rd level, you can press the `Overlay extra data` button to set two more levels of extra data into your query. Pressing `Set Query` button, apart from removing selected values, will show you a summary table with the constraints you have defined. If you have built a query that can be run against the InterMine instance you will be able to press in the lower right corner the `Go to Results` button that will take you to the next section.

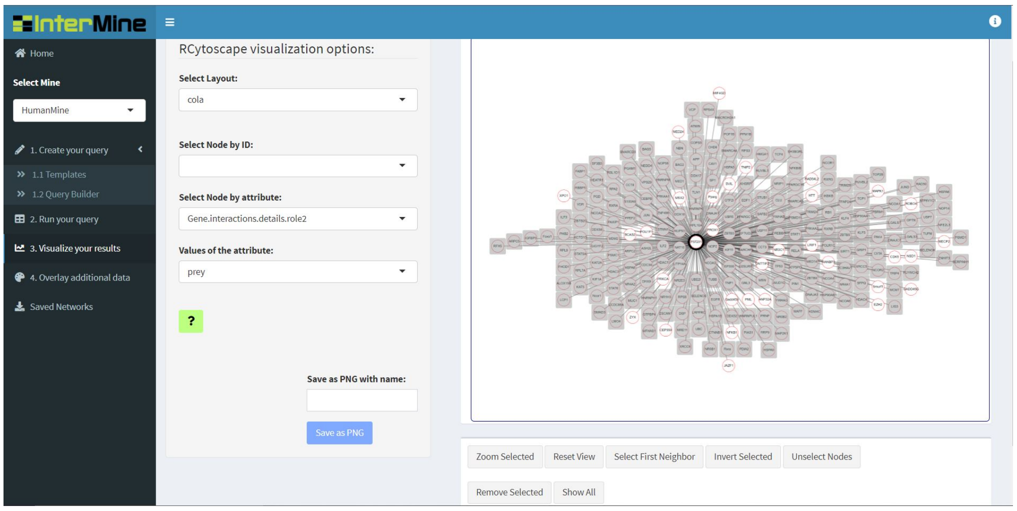
3.4 2. Run your query tab

In this section, a table containing the data which were retrieved from the InterMine instance is displayed. You will need to select the `Set Nodes and Edges` button to select the Id and the Source for the Cytoscape Network Visualization. The Id and Source need to be different. At this point, you may also want to define the `Node Attributes` to then be able to manipulate the network chart according to filtering criteria given by these attributes.

3.5 3. Visualize your results tab

Here, you can set different visualization and filter options. First, you can choose different layouts from the `Select Layout` list. Specific nodes from your network can be selected by attribute or ID, using buttons at the left of the page.

Buttons below the network allow various actions based on the nodes selected: remove selected (you can go back and display all the initial nodes pressing the `Show All` button), zoom selected, reset the view or select the first neighbour of a node or a selection of nodes. In addition you can invert the selection, unselect nodes and display a list of names from the selected nodes.



Select Mine: HumanMine

RCytoscape visualization options:

Select Layout: cola

Select Node by ID:

Select Node by attribute: Gene.interactions.details.role2

Values of the attribute: prey

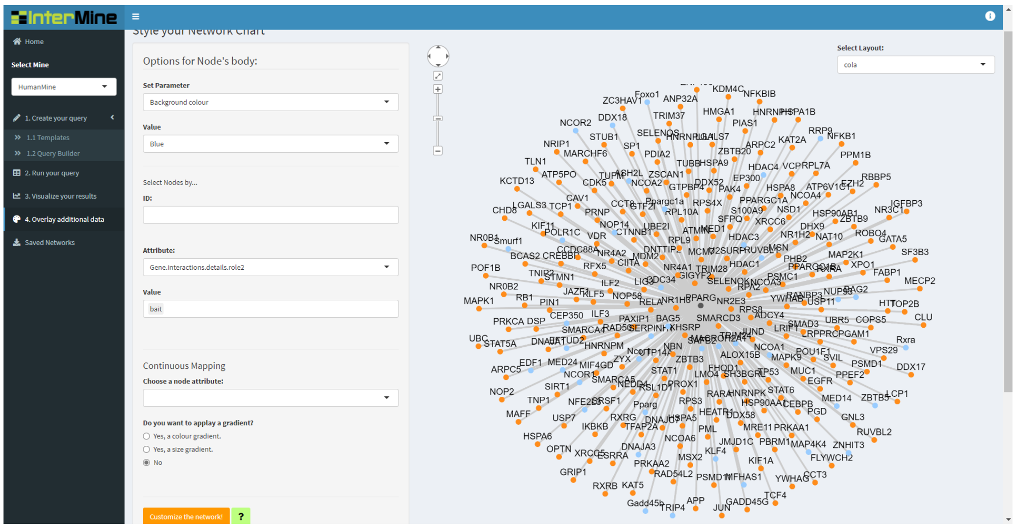
Figure 3: Caption of the Visualize your results tab.

3.6 4. Overlay additional data tab

In this last tab, you can easily style the network chart. In the Node's body menu first, select which parameter you want to customize using `Set Parameter`. Second, you need to define the value for the parameter. You can either set the node(s) to customize by their ID or by an attribute. You can select one of the attributes set in the Run Query Section to filter the customization. You must select the `Value` for the attribute if you have chosen this way. Each time you describe a new feature for the network do not forget to do double click to the `Customize the Network!` button. Pressing `History of changes` button you can see the changes you have made and delete some of them by pressing `Delete Rows`.

The newest feature added is the continuous-to-continuous mapping of attributes. You can apply a gradient, of node size or colour, to the node's attribute that you desired. With this feature, you will obtain nice network where numerical information is easily understood. You must select a node attribute from the ones chosen in the Run Query tab. Then, you can specify the range of values within the values of the attribute to map and choose between size or colour gradient. The size gradient can be set between 10 and 200 in pixels at zoom 1.

Finally, you can save your results as a static image, in PNG format, using



Style your Network Chart

Select Mine: HumanMine

Options for Node's body:

Set Parameter: Background colour

Value: Blue

Select Nodes by...

ID:

Attribute: Gene.interactions.details.role2

Value: bait

Continuous Mapping

Choose a node attribute:

Do you want to applay a gradient?

Yes, a colour gradient.

Yes, a size gradient.

[✓ No]

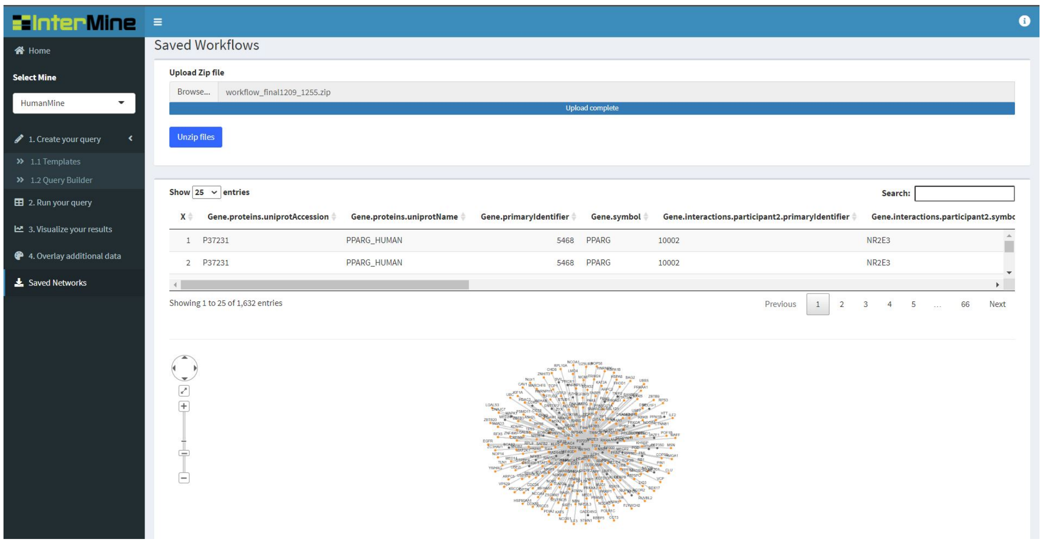
Select Layout: cola

Figure 4: Caption of the Overlay additional data tab. The background colour of the nodes corresponding to genes that in the interaction have a prey role is set to orange and for the ones with a bait role to blue.

the `Save as PNG` option. `Saved Networks` can be uploaded in the Saved Networks tab. If no name is provided for the image a default name will be given. In addition a ZIP folder with the files for an interactive network display can be saved using the `Save as ZIP` option in the Style your network charts tab. Again, if no name is provided a default name will be given. The ZIP folder will contain the results of the query in a CSV file, two CSV files with the basic components and the modifications made to the Network and a JSON file of the Network.

3.7 Saved Networks tab

In this last tab, you can display previously saved networks. You will need to press `Browse...` and navigate to find the zip folder you have saved in the tab `Overlay additional data`. Then, you will need to press `Unzip files` and the results of the query in a table and the network chart will be displayed.



workflow_final1209_1255.zip

X	Gene.proteins.uniprotAccession	Gene.proteins.uniprotName	Gene.primaryIdentifier	Gene.symbol	Gene.interactions.participant2.primaryIdentifier	Gene.interactions.participant2.symbc
1	P37231	PPARG_HUMAN	5468	PPARG	10002	NR2E3
2	P37231	PPARG_HUMAN	5468	PPARG	10002	NR2E3

Showing 1 to 25 of 1,632 entries

Figure 5: Caption of the Saved Networks tab displaying the example Network chart shown in Figure 4.

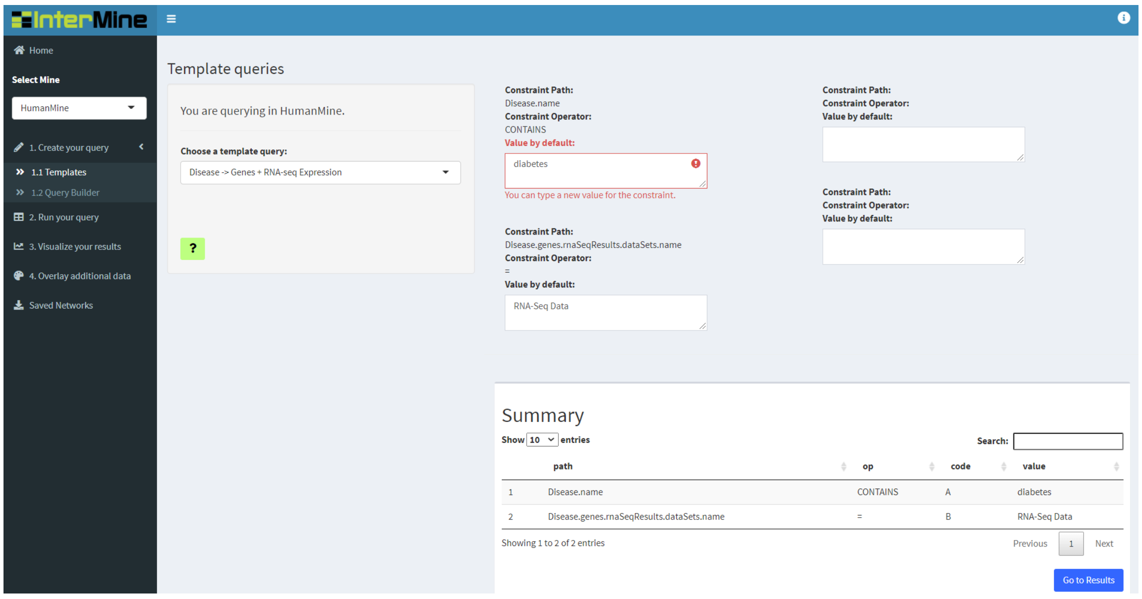
4 Source Code

InterMineR Cytoscape Interface is open source and may be downloaded and forked on Github. Pull Requests are welcomed!

APPENDIX C. USE-CASES FOR THE INTERMINER-CYTOSCAPE SHINY INTERFACE

C.1. HumanMine use-case.

C.1.1. Workflow A:



Template queries

You are querying in HumanMine.

Choose a template query: Disease -> Genes + RNA-seq Expression

Constraint Path: Disease.name
Constraint Operator: CONTAINS
Value by default: diabetes
You can type a new value for the constraint.

Constraint Path: Disease.genes.rnaSeqResults.dataSets.name
Constraint Operator: =
Value by default: RNA-Seq Data

Constraint Path:
Constraint Operator:
Value by default:

Constraint Path:
Constraint Operator:
Value by default:

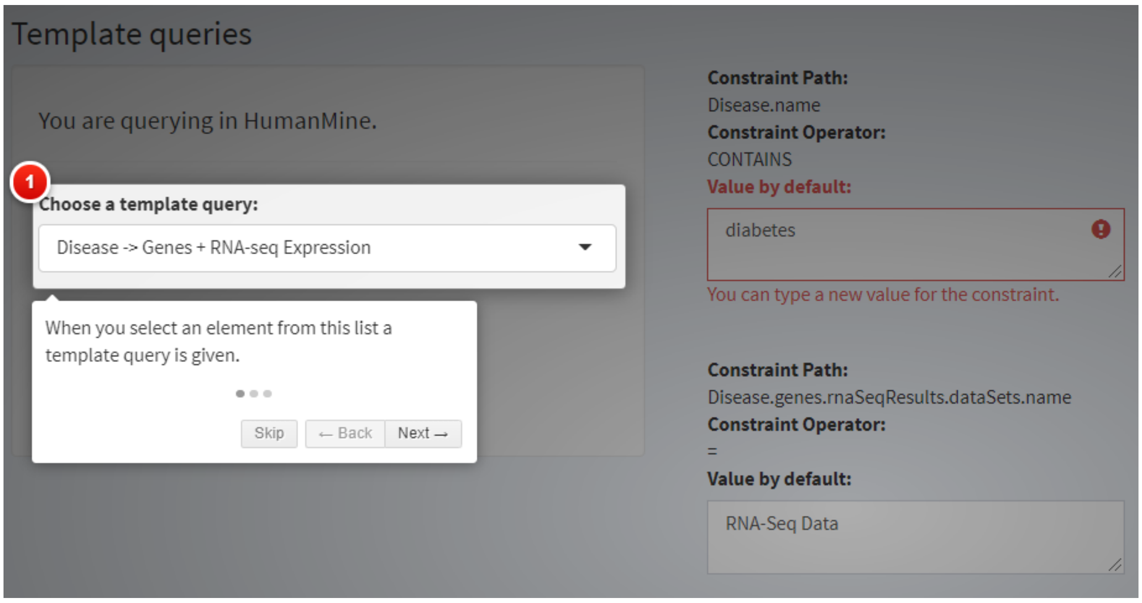
Summary

Show 10 entries — Search:

	path	op	code	value
1	Disease.name	CONTAINS	A	diabetes
2	Disease.genes.rnaSeqResults.dataSets.name	=	B	RNA-Seq Data

Showing 1 to 2 of 2 entries — Previous 1 Next

Figure C.1: Global view of the Template Queries tab.



Template queries

You are querying in HumanMine.

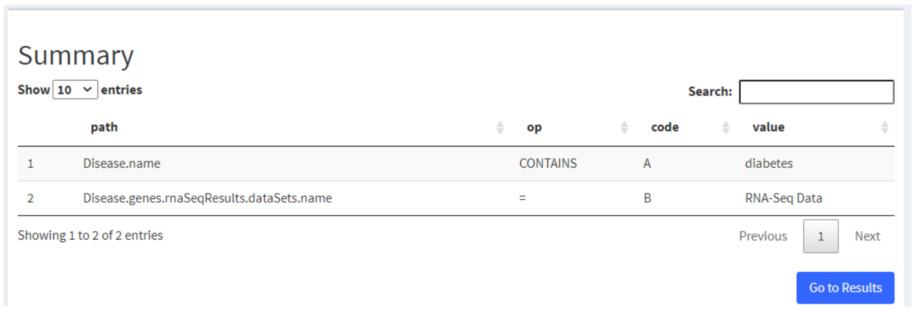
Choose a template query: Disease -> Genes + RNA-seq Expression

When you select an element from this list a template query is given.

Constraint Path: Disease.name
Constraint Operator: CONTAINS
Value by default: diabetes
You can type a new value for the constraint.

Constraint Path: Disease.genes.rnaSeqResults.dataSets.name
Constraint Operator: =
Value by default: RNA-Seq Data

Figure C.2: Choosing the Template Query.



Summary

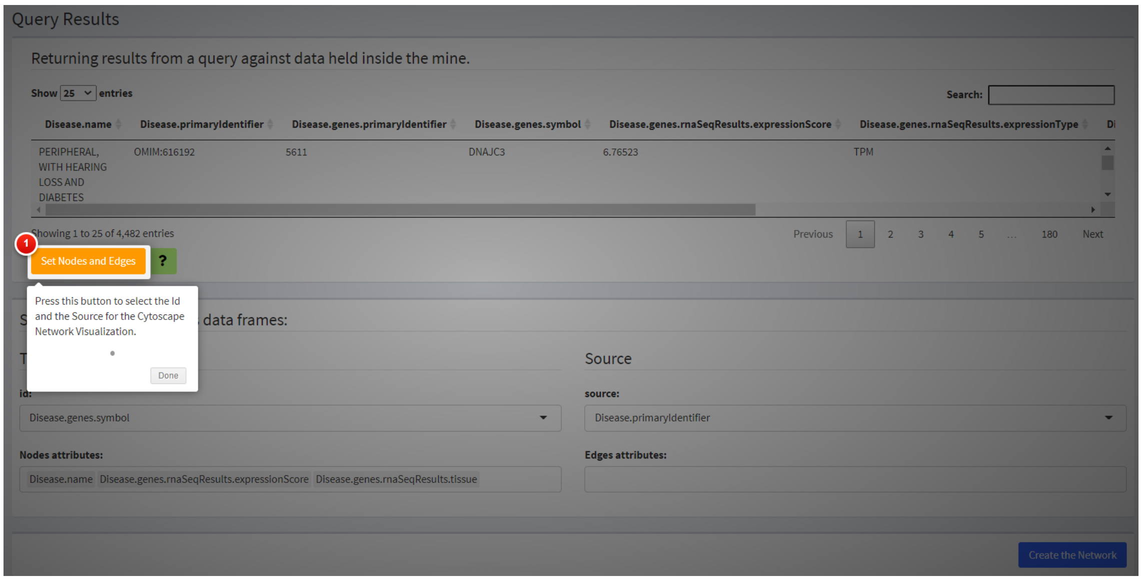
Show 10 entries

Search:

	path	op	code	value
1	Disease.name	CONTAINS	A	diabetes
2	Disease.genes.rnaSeqResults.dataSets.name	=	B	RNA-Seq Data

Showing 1 to 2 of 2 entries

Figure C.3: Summary of the constraints defined in the Template Query.



Query Results

Returning results from a query against data held inside the mine.

Show 25 entries

Search:

Disease.name	Disease.primaryIdentifier	Disease.genes.primaryIdentifier	Disease.genes.symbol	Disease.genes.rnaSeqResults.expressionScore	Disease.genes.rnaSeqResults.expressionType	Di
PERIPHERAL, WITH HEARING LOSS AND DIABETES	OMIM:616192	5611	DNAJC3	6.76523	TPM	

Showing 1 to 25 of 4,482 entries

Press this button to select the Id and the Source for the Cytoscape Network Visualization.

S data frames:

T

Source

id: Disease.genes.symbol

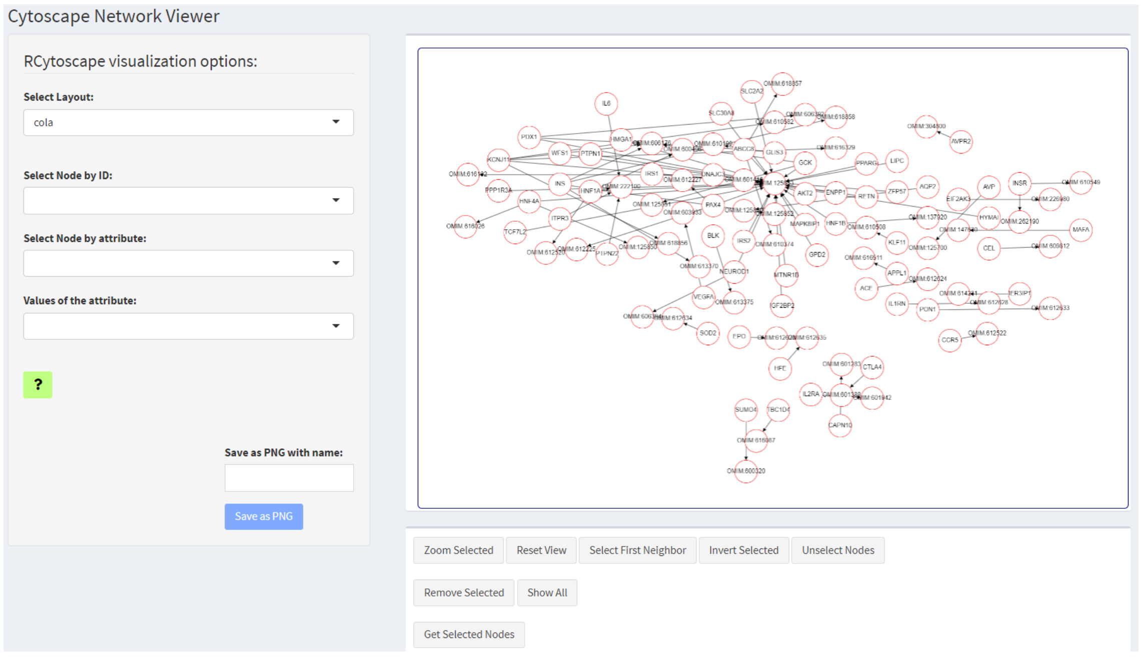
source: Disease.primaryIdentifier

Nodes attributes: Disease.name Disease.genes.rnaSeqResults.expressionScore Disease.genes.rnaSeqResults.tissue

Edges attributes:

Figure C.4: Table of results and selection of nodes, edges and nodes' attributes.

C.1.2. Workflow B:


Cytoscape Network Viewer

RCytoscape visualization options:

Select Layout: cola

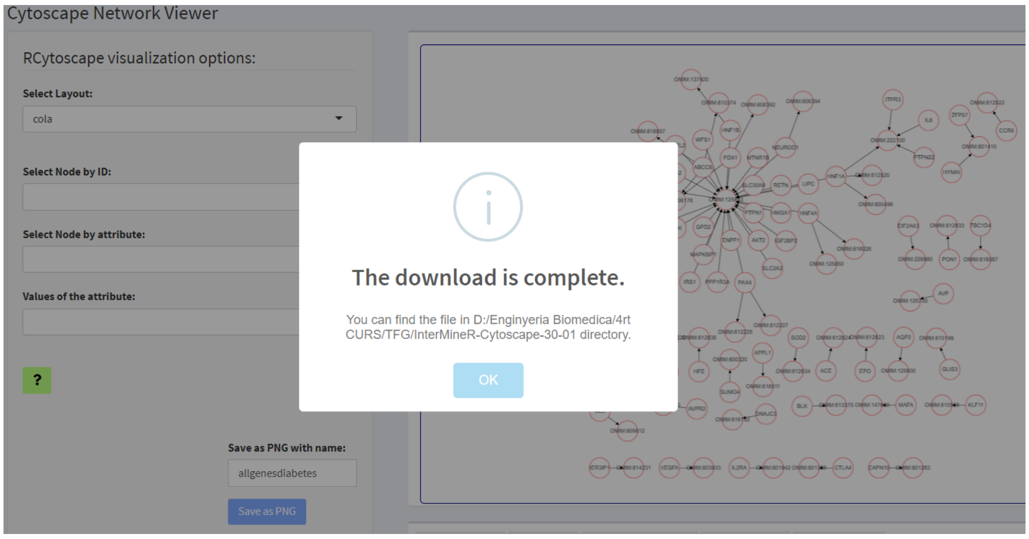
Select Node by ID:

Select Node by attribute:

Values of the attribute:

Save as PNG with name:

Figure C.5: Initial view of the Visualize your results tab.


Cytoscape Network Viewer

RCytoscape visualization options:

Select Layout: cola

Select Node by ID:

Select Node by attribute:

Values of the attribute:

Save as PNG with name: allgenesdiabetes

The download is complete.

You can find the file in D:/Enginyeria Biomedica/4rt CURS/TFG/InterMineR-Cytoscape-30-01 directory.

Figure C.6: Cola layout and saving an image of the entire network.

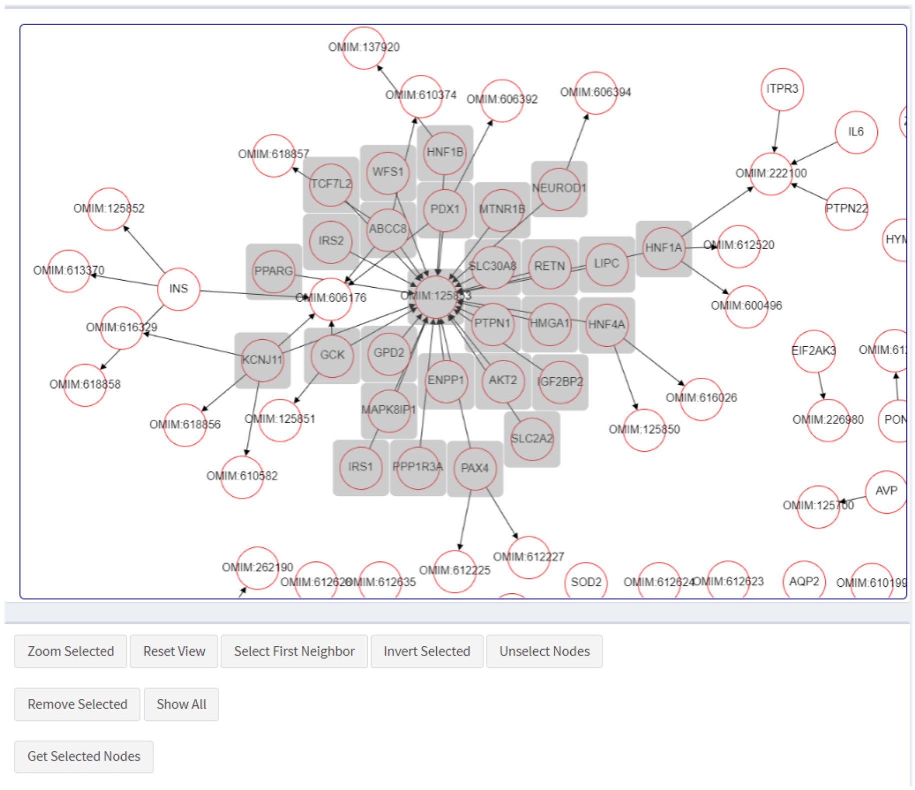


Figure C.7: "Zoom selected" view of the node *OMIM:125853* and first neighbours.

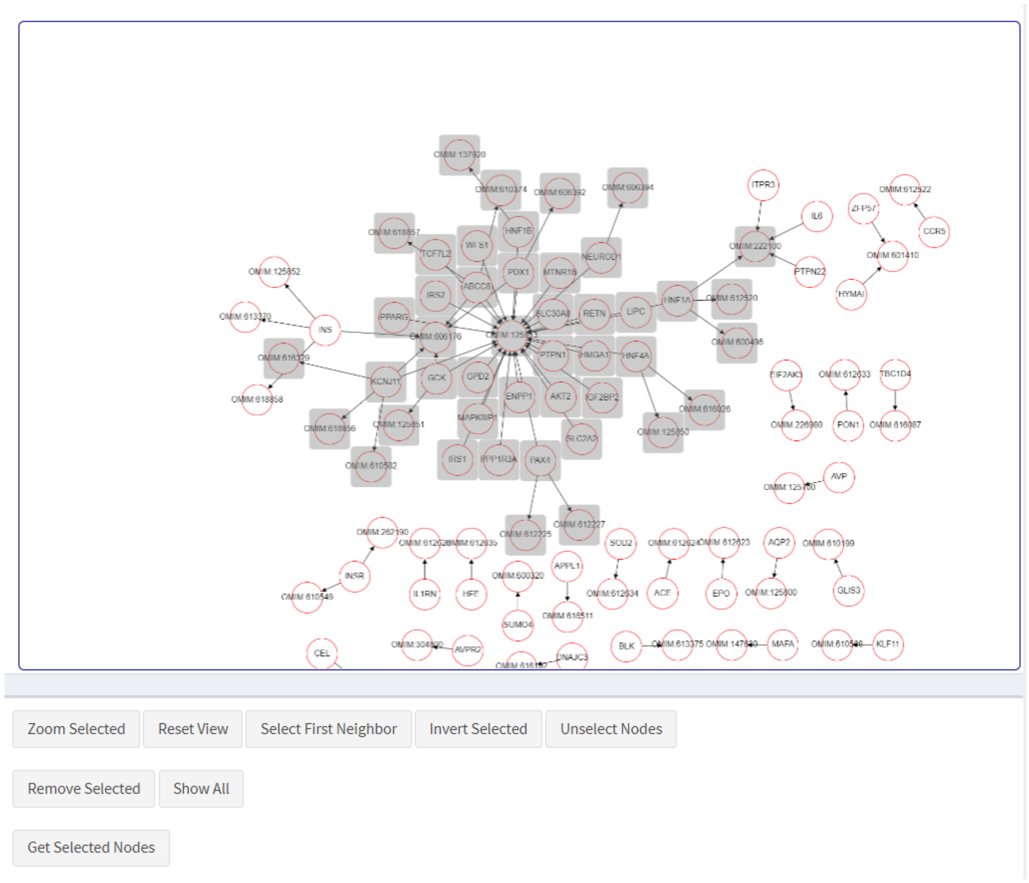


Figure C.8: "Zoom selected" view of the first neighbours of the previous selection C.7.

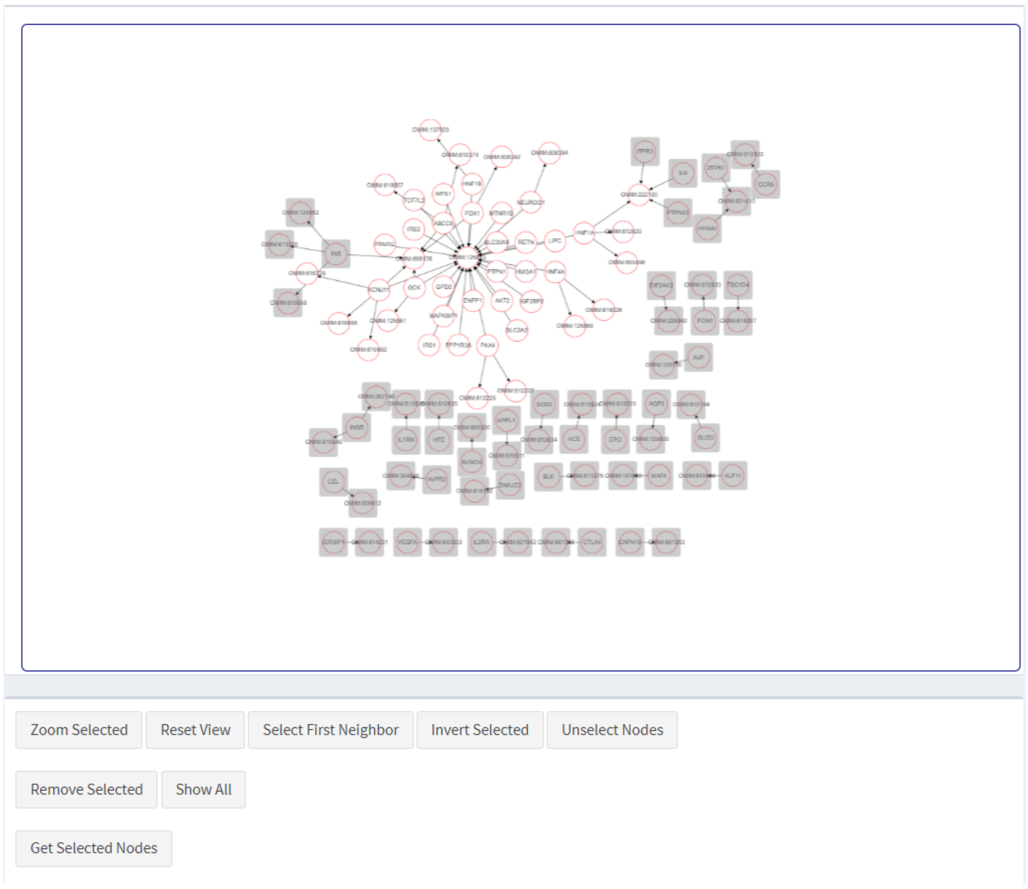


Figure C.9: Invert selected of the previous selection C.8.

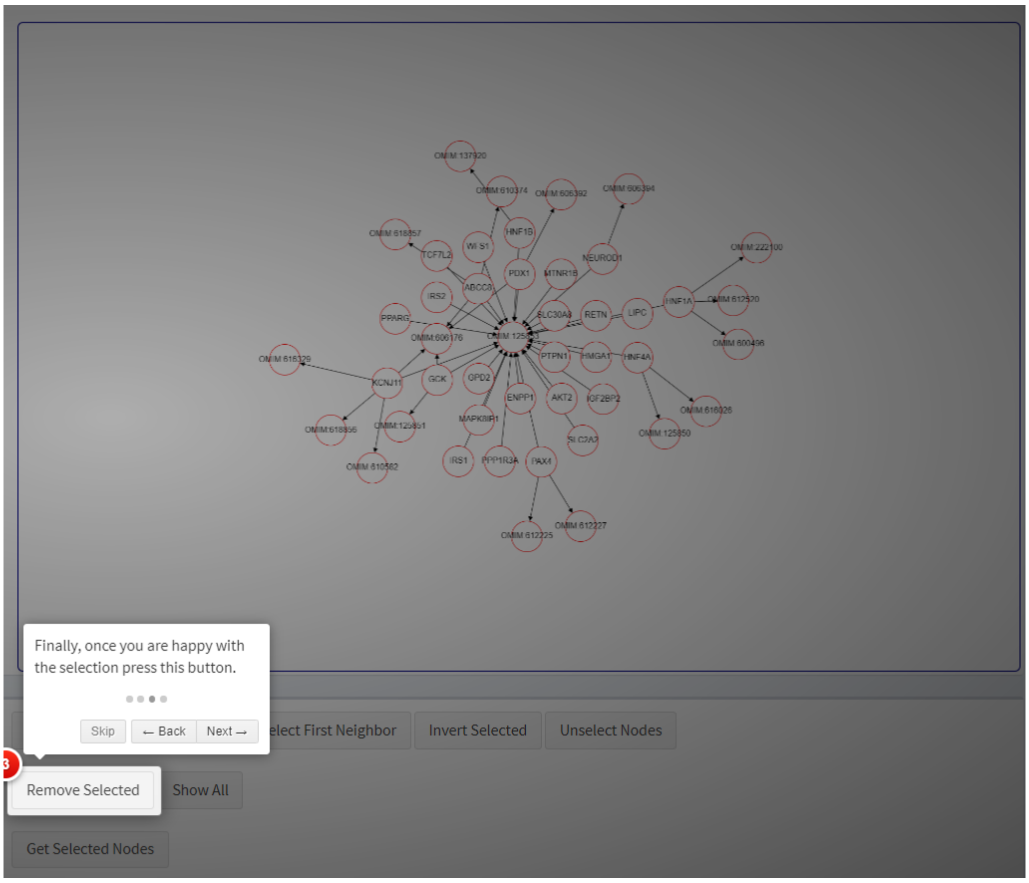


Figure C.10: Remove selected of the previous selection C.9.

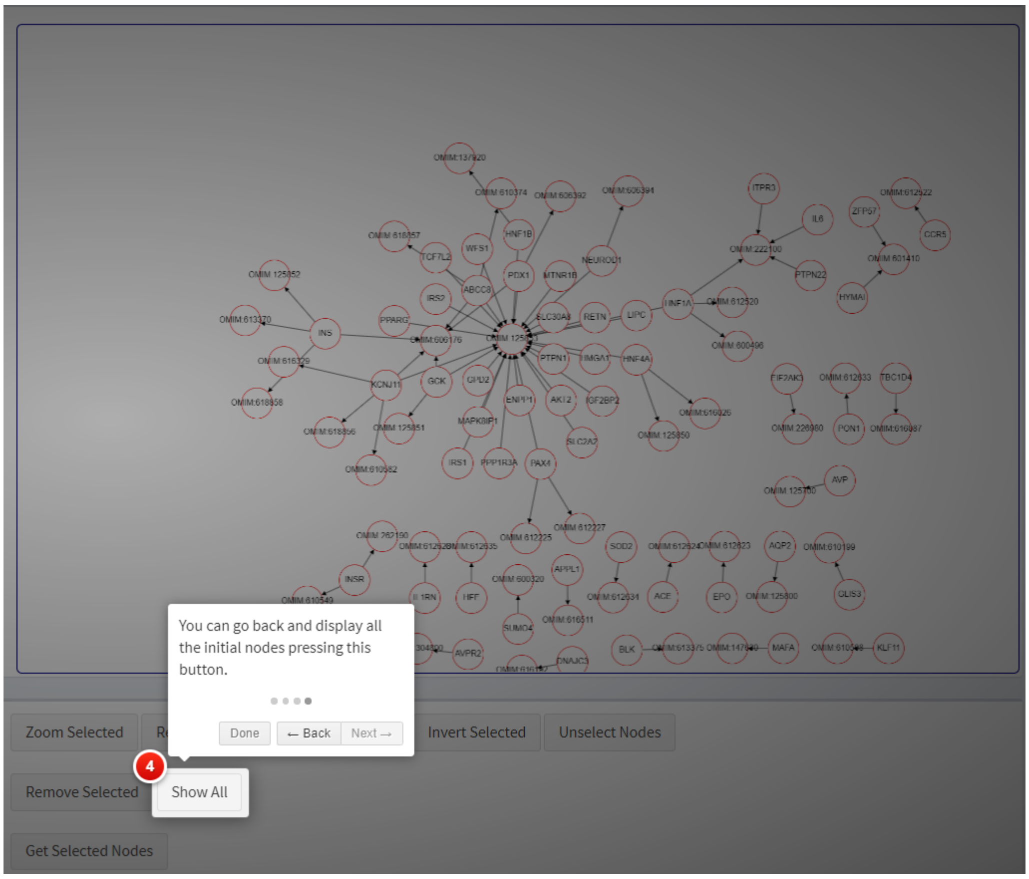
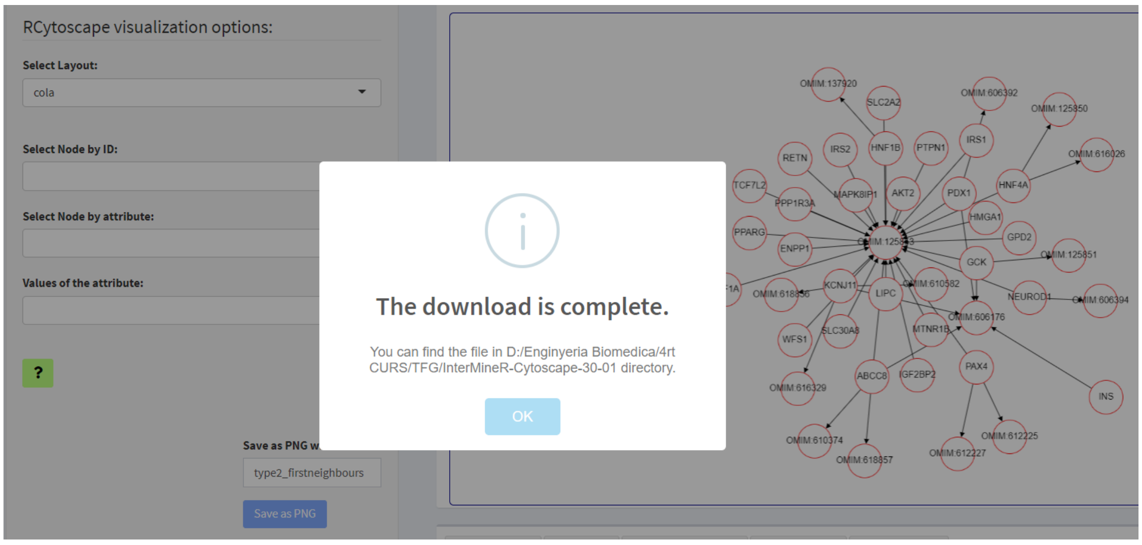

You can go back and display all the initial nodes pressing this button.

Figure C.11: Show all the nodes.


RCytoscape visualization options:

Select Layout: cola

Select Node by ID:

Select Node by attribute:

Values of the attribute:

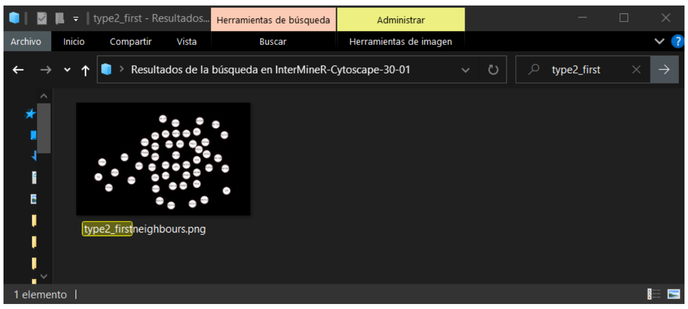
Save as PNG w

type2_firstneighbours

The download is complete.

You can find the file in D:/Enginyeria Biomedica/4rt CURS/TFG/InterMineR-Cytoscape-30-01 directory.

Figure C.12: First neighbours of *OMIM:125853* network saved as an image.


type2_first - Resultados... Herramientas de búsqueda Administrar

Archivo Inicio Compartir Vista Buscar Herramientas de imagen

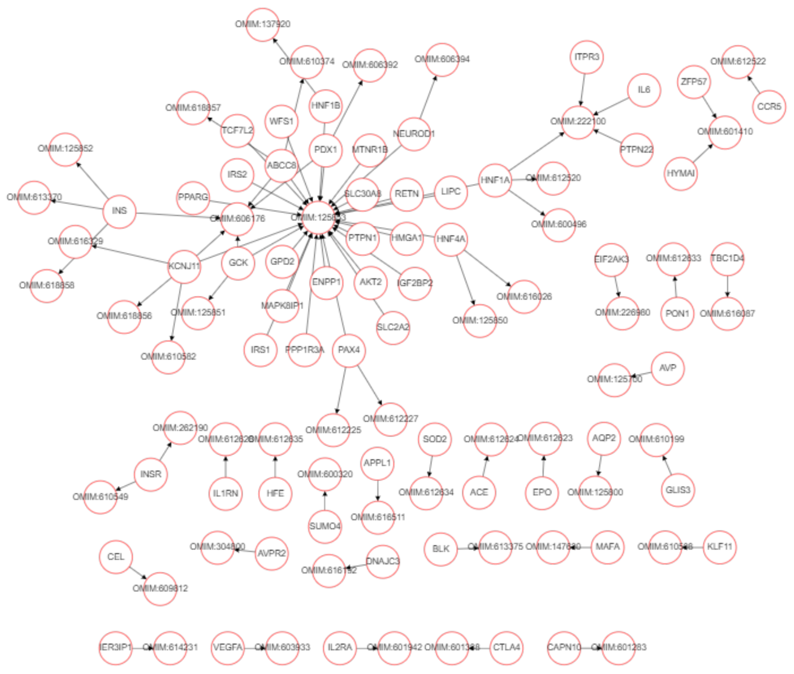
Resultados de la búsqueda en InterMineR-Cytoscape-30-01

type2_first

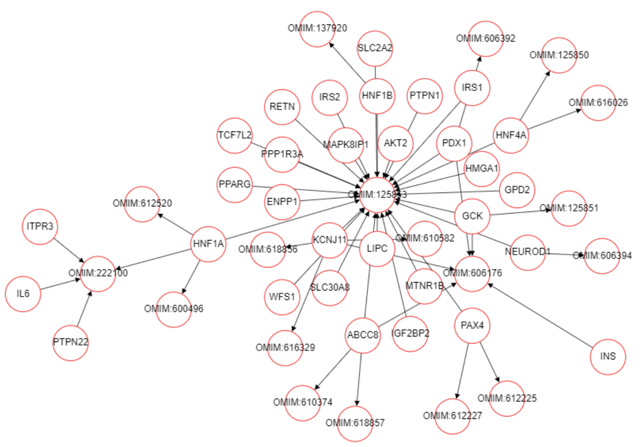
type2_firstneighbours.png

1 elemento

Figure C.13: The directory where the image C.12 has been saved.



(a) Original.



(b) Filtered by First Neighbours.

Figure C.14: Networks of the Workflow A.

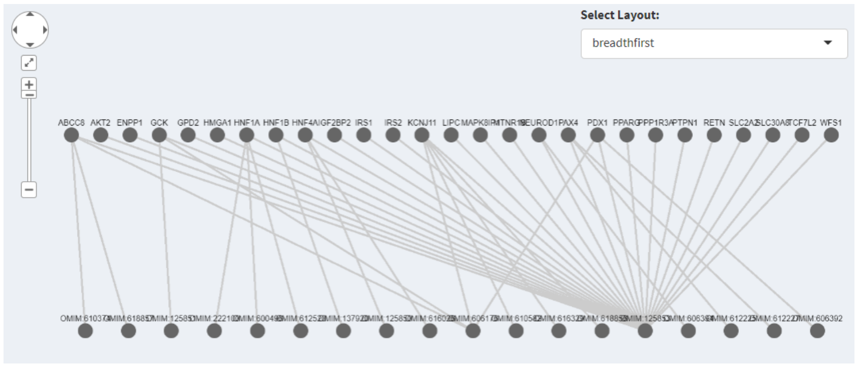
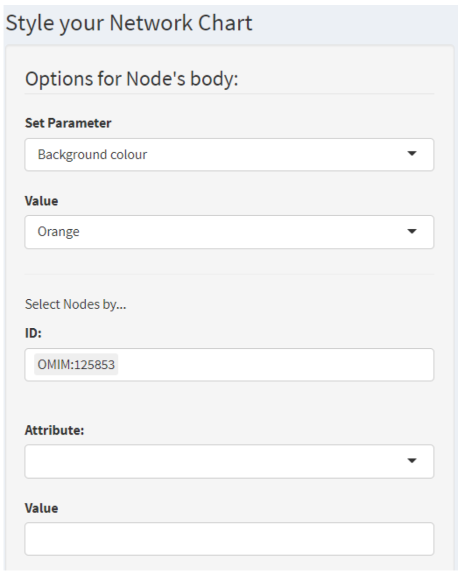


Figure C.15: Initial view of the Overlay additional data tab.



Style your Network Chart

Options for Node's body:

Set Parameter: Background colour

Value: Orange

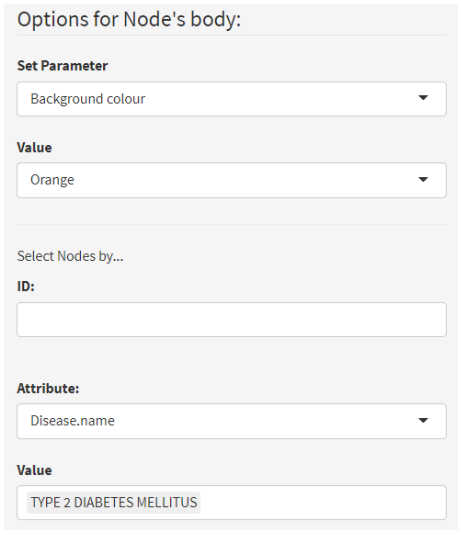
Select Nodes by...

ID: OMIM:125853

Attribute:

Value:

(a) Background-colour by ID.



Options for Node's body:

Set Parameter: Background colour

Value: Orange

Select Nodes by...

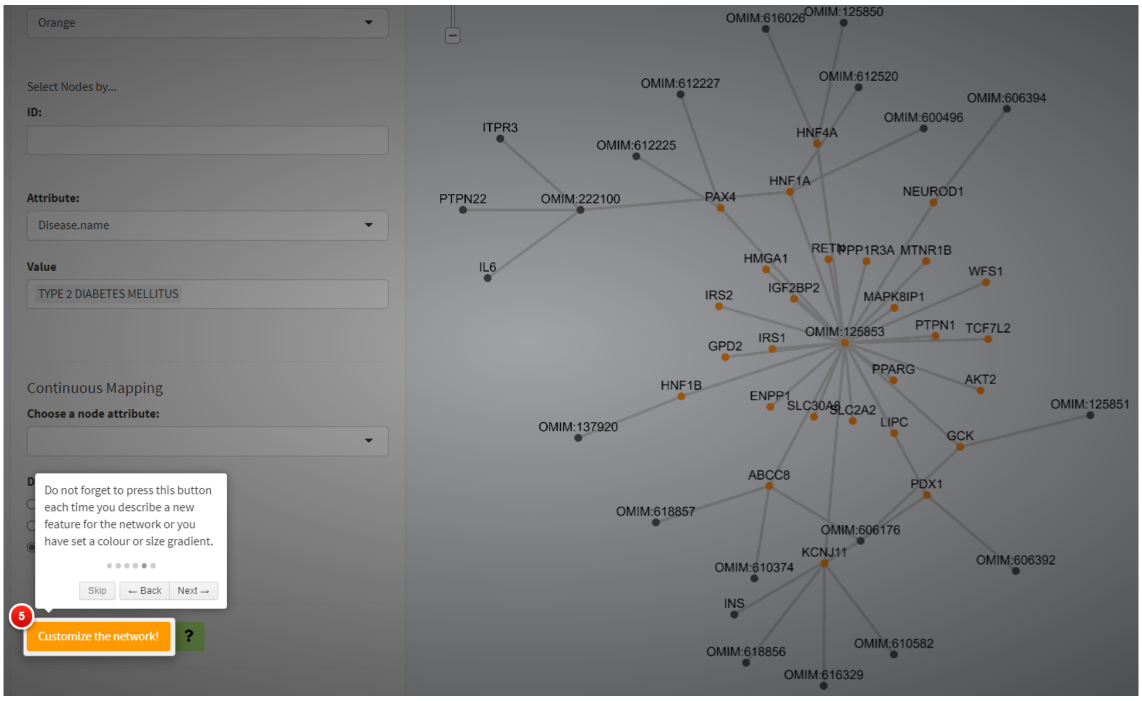
ID:

Attribute: Disease.name

Value: TYPE 2 DIABETES MELLITUS

(b) Background-color by attribute.

Figure C.16: Customization of the genes related with Diabetes Mellitus Type 2.



Orange

Select Nodes by...

ID:

Attribute:

Disease.name

Value

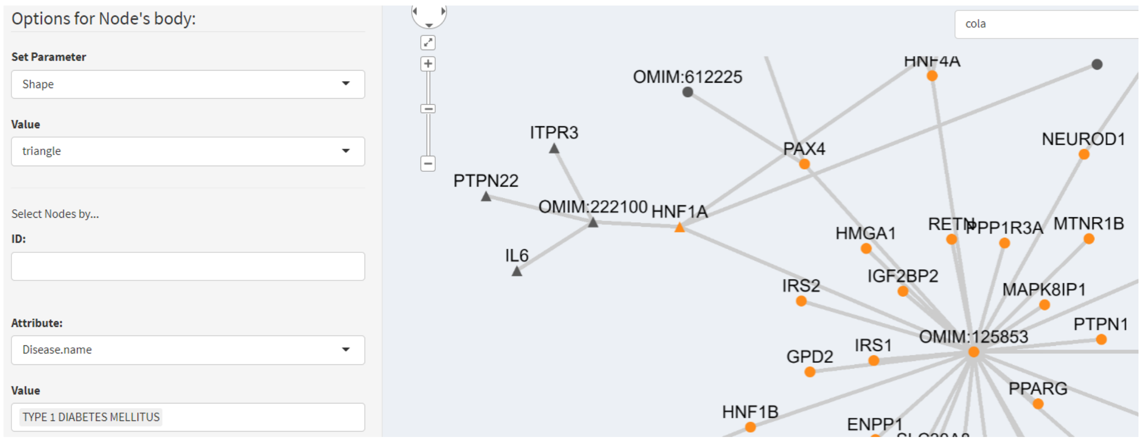
TYPE 2 DIABETES MELLITUS

Continuous Mapping

Choose a node attribute:

Do not forget to press this button each time you describe a new feature for the network or you have set a colour or size gradient.

Figure C.17: Results of the orange background-colour filter C.16.



Options for Node's body:

Set Parameter

Shape

Value

triangle

Select Nodes by...

ID:

Attribute:

Disease.name

Value

TYPE 1 DIABETES MELLITUS

Figure C.18: Customization of the genes related with Diabetes Mellitus Type 1.

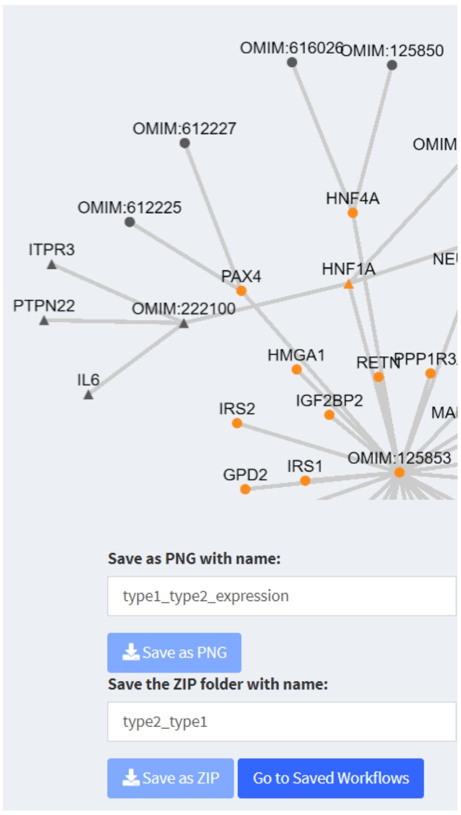
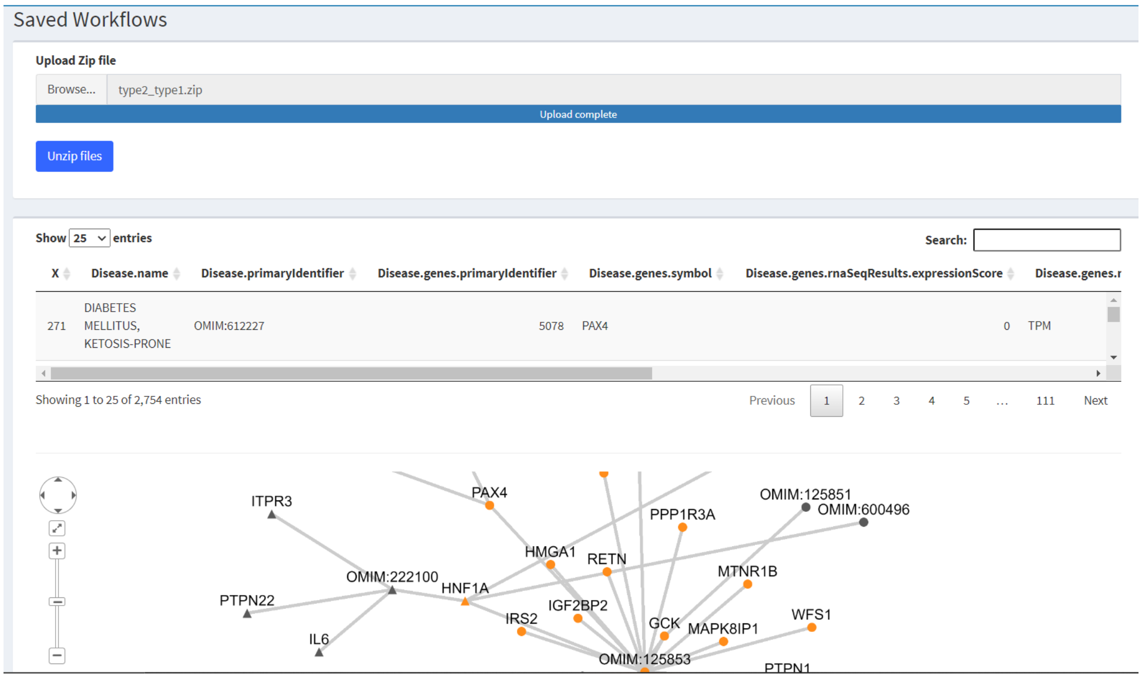


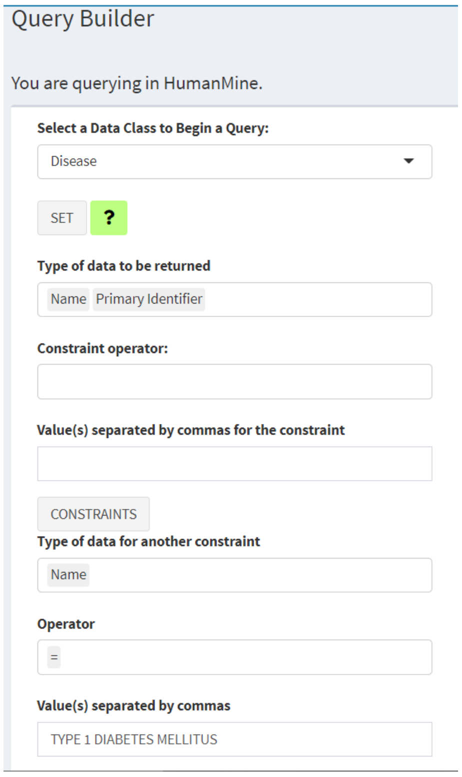
Figure C.19: Saving the customized network.



X	Disease.name	Disease.primaryIdentifier	Disease.genes.primaryIdentifier	Disease.genes.symbol	Disease.genes.rnaSeqResults.expressionScore	Disease.genes.r
271	DIABETES MELLITUS, KETOSIS-PRONE	OMIM:612227	5078	PAX4	0	TPM

Showing 1 to 25 of 2,754 entries

Figure C.20: Displaying the saved network C.19 in the Saved Workflows tab.



Query Builder

You are querying in HumanMine.

Select a Data Class to Begin a Query:

Disease

Type of data to be returned

Name Primary Identifier

Constraint operator:

Value(s) separated by commas for the constraint

Type of data for another constraint

Name

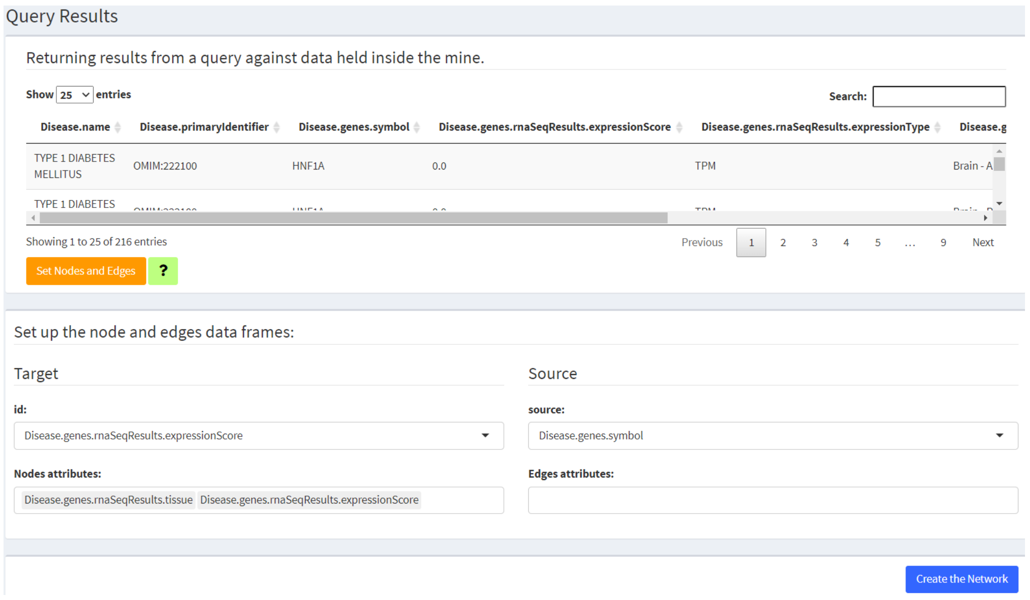
Operator

=

Value(s) separated by commas

TYPE 1 DIABETES MELLITUS

Figure C.21: Query Builder view of first level data class and the constraint for *Disease.name*.



Query Results

Returning results from a query against data held inside the mine.

Show 25 entries — Search:

Disease.name	Disease.primaryIdentifier	Disease.genes.symbol	Disease.genes.rnaSeqResults.expressionScore	Disease.genes.rnaSeqResults.expressionType	Disease.g
TYPE 1 DIABETES MELLITUS	OMIM:222100	HNF1A	0.0	TPM	Brain - A
TYPE 1 DIABETES	OMIM:222100	HNF1A	0.0	TPM	Brain - D

Showing 1 to 25 of 216 entries

Set up the node and edges data frames:

Target

id: Disease.genes.rnaSeqResults.expressionScore

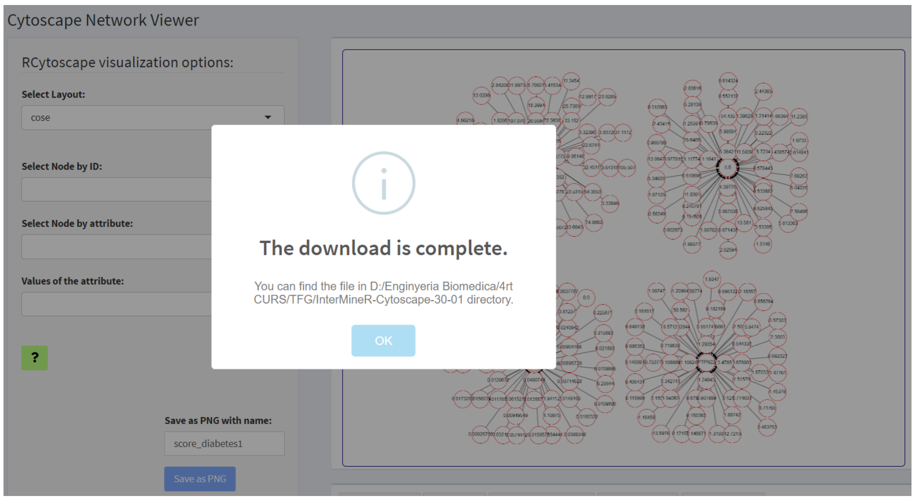
Nodes attributes: Disease.genes.rnaSeqResults.tissue Disease.genes.rnaSeqResults.expressionScore

Source

source: Disease.genes.symbol

Edges attributes:

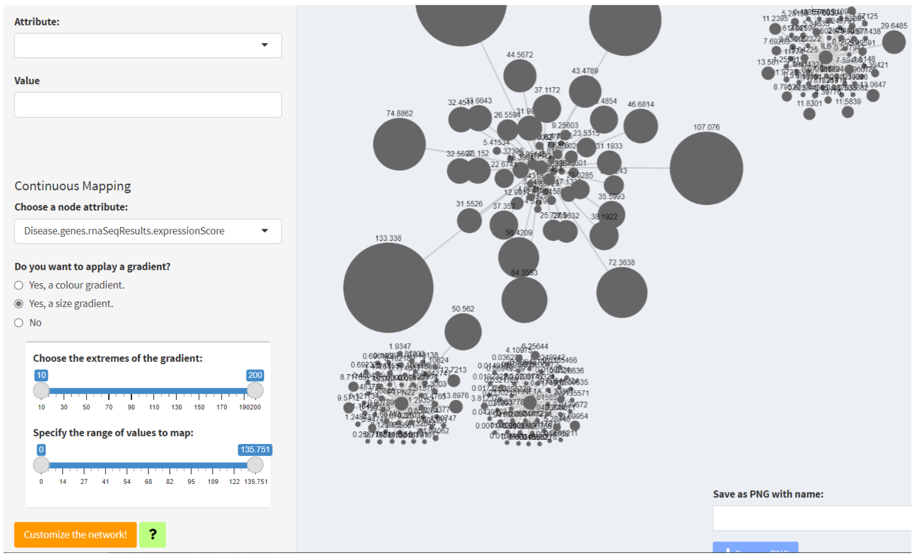
Figure C.22: Query Results view and selection of target and source data.



The download is complete.

You can find the file in D:/Enginyeria Biomedica/4rt CURS/TFG/InterMineR-Cytoscape-30-01 directory.

Figure C.23: Cytoscape Network Viewer of the results.



Choose a node attribute: Disease.genes.rnaSeqResults.expressionScore

Do you want to applay a gradient?

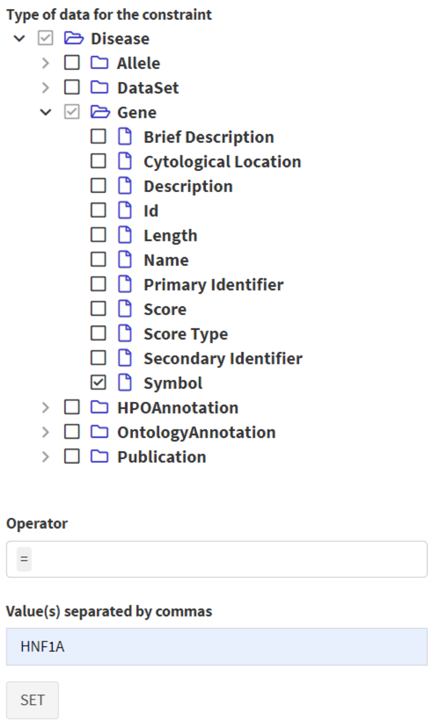
- ( ) Yes, a colour gradient.
- [✓ Yes, a size gradient.]
- ( ) No

Choose the extremes of the gradient: 10 – 200

Specify the range of values to map: 0 – 135.751

Figure C.24: Size gradient of expression score for the genes expressed in Diabetes Mellitus Type 1.

C.1.2.1. Gene HNF1A:



Type of data for the constraint

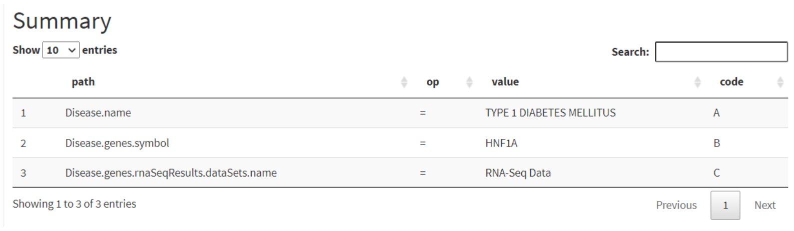
- ☑ Disease
  - ☐ Allele
  - ☐ DataSet
  - ☑ Gene
    - ☐ Brief Description
    - ☐ Cytological Location
    - ☐ Description
    - ☐ Id
    - ☐ Length
    - ☐ Name
    - ☐ Primary Identifier
    - ☐ Score
    - ☐ Score Type
    - ☐ Secondary Identifier
    - ☑ Symbol
  - ☐ HPOAnnotation
  - ☐ OntologyAnnotation
  - ☐ Publication

Operator: =

Value(s) separated by commas: HNF1A

SET

Figure C.25: Second constraint in the Query Builder for *Gene.symbol*.



Summary

Show 10 entries

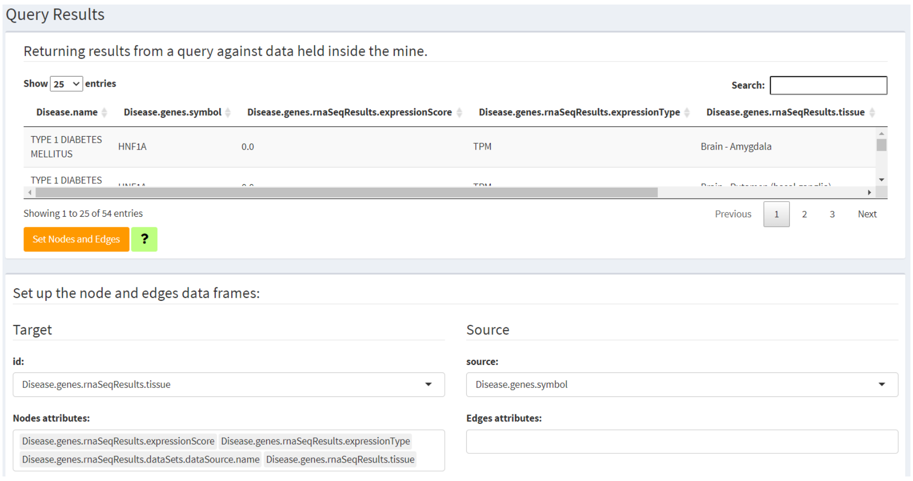
Search:

	path	op	value	code
1	Disease.name	=	TYPE 1 DIABETES MELLITUS	A
2	Disease.genes.symbol	=	HNF1A	B
3	Disease.genes.rnaSeqResults.dataSets.name	=	RNA-Seq Data	C

Showing 1 to 3 of 3 entries

Previous 1 Next

Figure C.26: Summary of constraints.



Query Results

Returning results from a query against data held inside the mine.

Show 25 entries | Search:

Disease.name	Disease.genes.symbol	Disease.genes.rnaSeqResults.expressionScore	Disease.genes.rnaSeqResults.expressionType	Disease.genes.rnaSeqResults.tissue
TYPE 1 DIABETES MELLITUS	HNF1A	0.0	TPM	Brain - Amygdala
TYPE 1 DIABETES				

Showing 1 to 25 of 54 entries

Set up the node and edges data frames:

Target

id: Disease.genes.rnaSeqResults.tissue

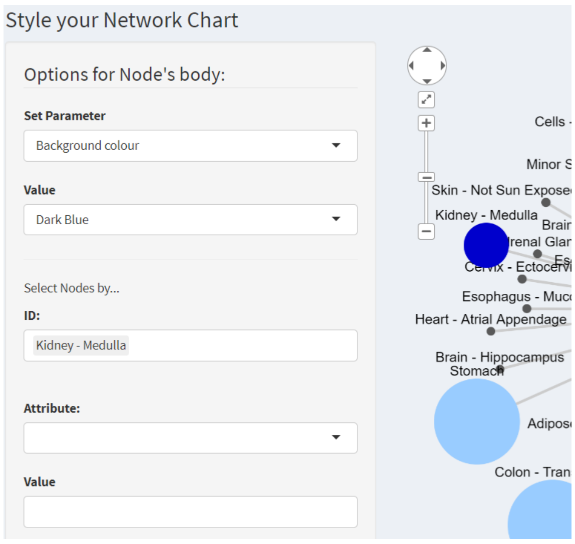
Nodes attributes: Disease.genes.rnaSeqResults.expressionScore Disease.genes.rnaSeqResults.expressionType Disease.genes.rnaSeqResults.dataSets.dataSource.name Disease.genes.rnaSeqResults.tissue

Source

source: Disease.genes.symbol

Edges attributes:

Figure C.27: Query Results view and selection of target and source data.



Style your Network Chart

Options for Node's body:

Set Parameter: Background colour

Value: Dark Blue

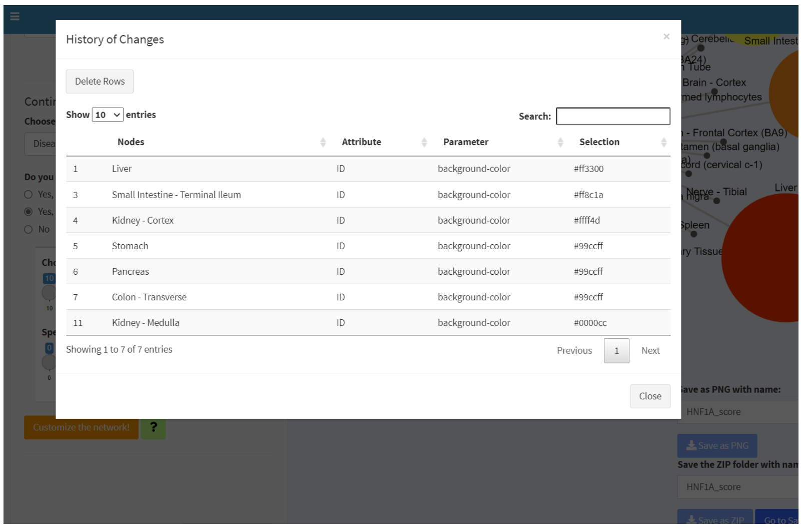
Select Nodes by...

ID: Kidney - Medulla

Attribute:

Value:

Figure C.28: Background-colour overlaying.



History of Changes

Show 10 entries | Search:

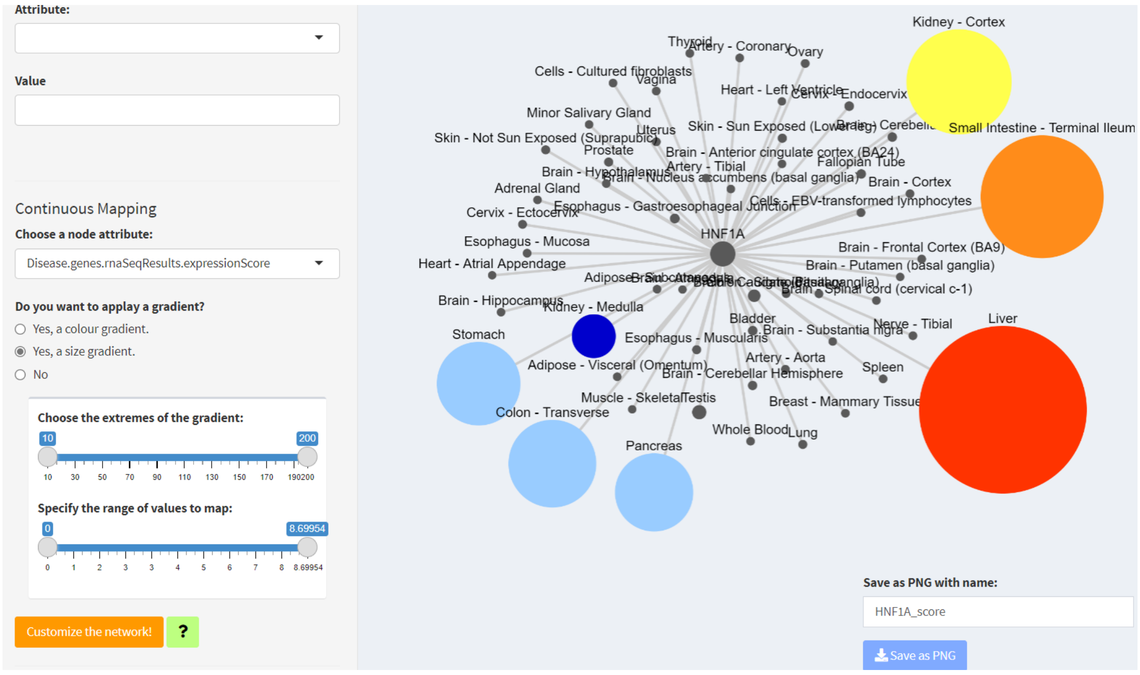
	Nodes	Attribute	Parameter	Selection
1	Liver	ID	background-color	#ff3300
3	Small Intestine - Terminal Ileum	ID	background-color	#ff8c1a
4	Kidney - Cortex	ID	background-color	#ffff4d
5	Stomach	ID	background-color	#99ccff
6	Pancreas	ID	background-color	#99ccff
7	Colon - Transverse	ID	background-color	#99ccff
11	Kidney - Medulla	ID	background-color	#0000cc

Showing 1 to 7 of 7 entries

Save as PNG with name: HNF1A_score

Save the ZIP folder with nam HNF1A_score

Figure C.29: History of Changes.



Attribute:

Value

Continuous Mapping

Choose a node attribute: Disease.genes.rnaSeqResults.expressionScore

Do you want to apply a gradient?

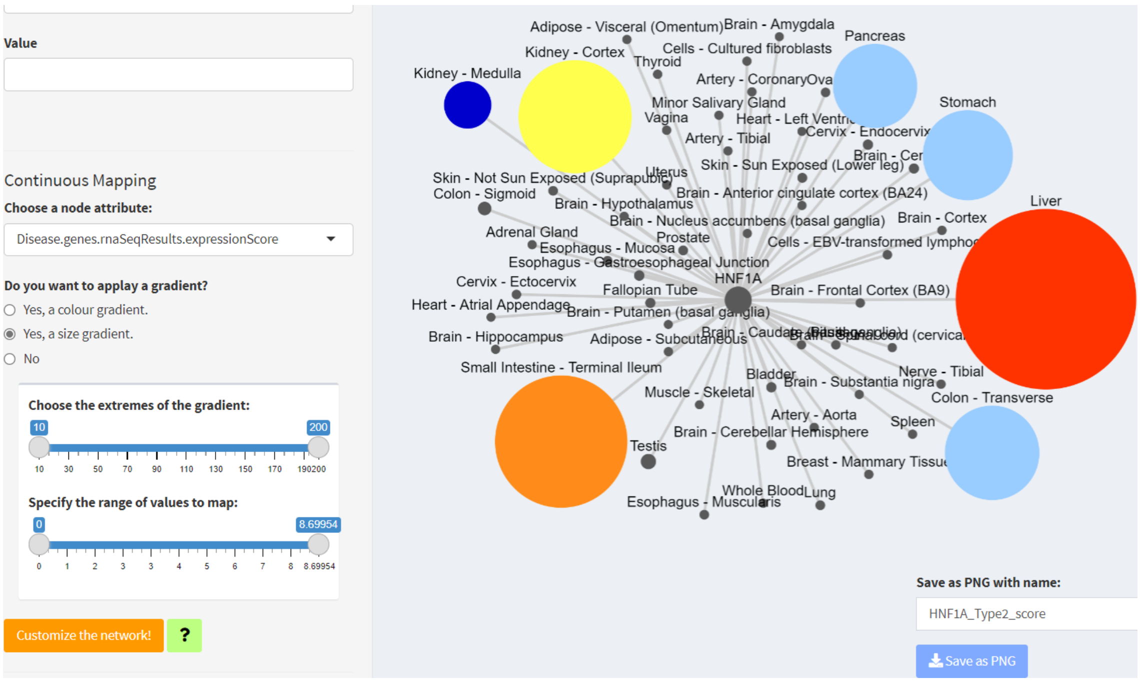
Yes, a colour gradient. [✓ Yes, a size gradient.] No

Choose the extremes of the gradient: 10 – 200

Specify the range of values to map: 0 – 8.69954

Save as PNG with name: HNF1A_score

(a) HNF1A in Diabetes Mellitus Type 1.



Value

Continuous Mapping

Choose a node attribute: Disease.genes.rnaSeqResults.expressionScore

Do you want to apply a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

Choose the extremes of the gradient: 10 – 200

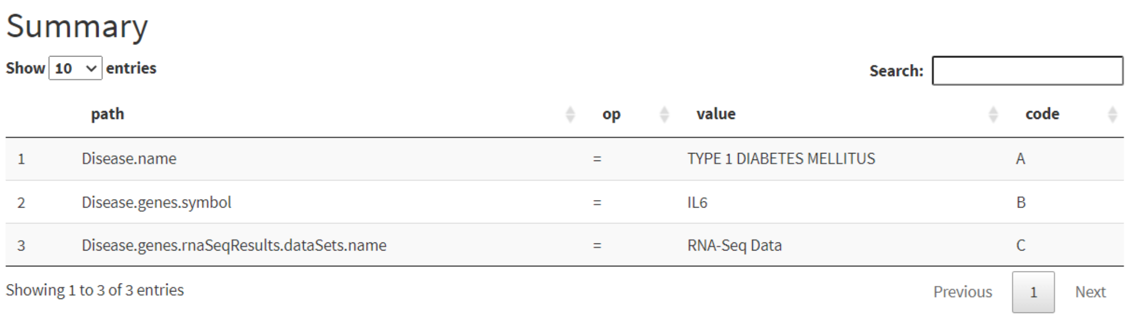
Specify the range of values to map: 0 – 8.69954

Save as PNG with name: HNF1A_Type2_score

(b) HNF1A in Diabetes Mellitus Type 2.

Figure C.30: Size gradient of expression score for different tissues.

C.1.2.2. Gene IL6:



Summary

Show 10 entries

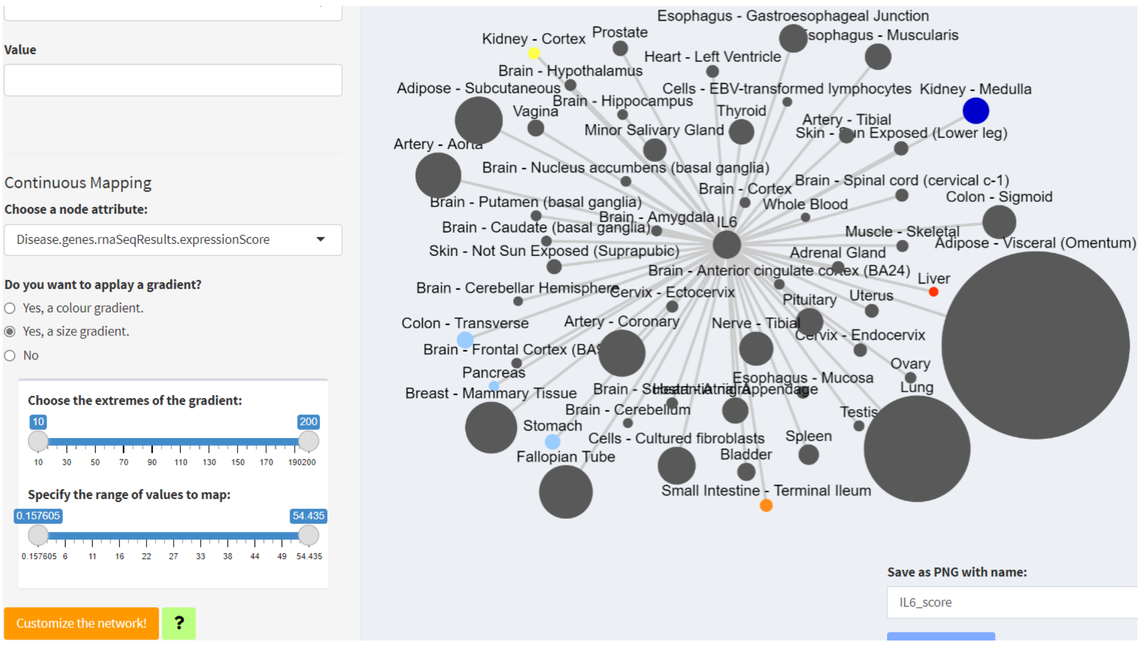
Search:

	path	op	value	code
1	Disease.name	=	TYPE 1 DIABETES MELLITUS	A
2	Disease.genes.symbol	=	IL6	B
3	Disease.genes.rnaSeqResults.dataSets.name	=	RNA-Seq Data	C

Showing 1 to 3 of 3 entries

Previous 1 Next

Figure C.31: Summary of constraints.



Value

Continuous Mapping

Choose a node attribute: Disease.genes.rnaSeqResults.expressionScore

Do you want to apply a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

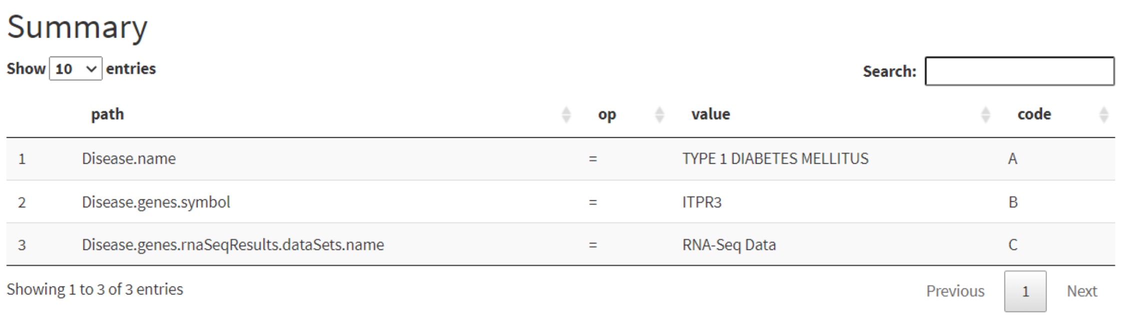
Choose the extremes of the gradient: 10 – 200

Specify the range of values to map: 0.157605 – 54.435

Save as PNG with name: IL6_score

Figure C.32: Size gradient of expression score for the IL6 gene.

C.1.2.3. Gene ITPR3:



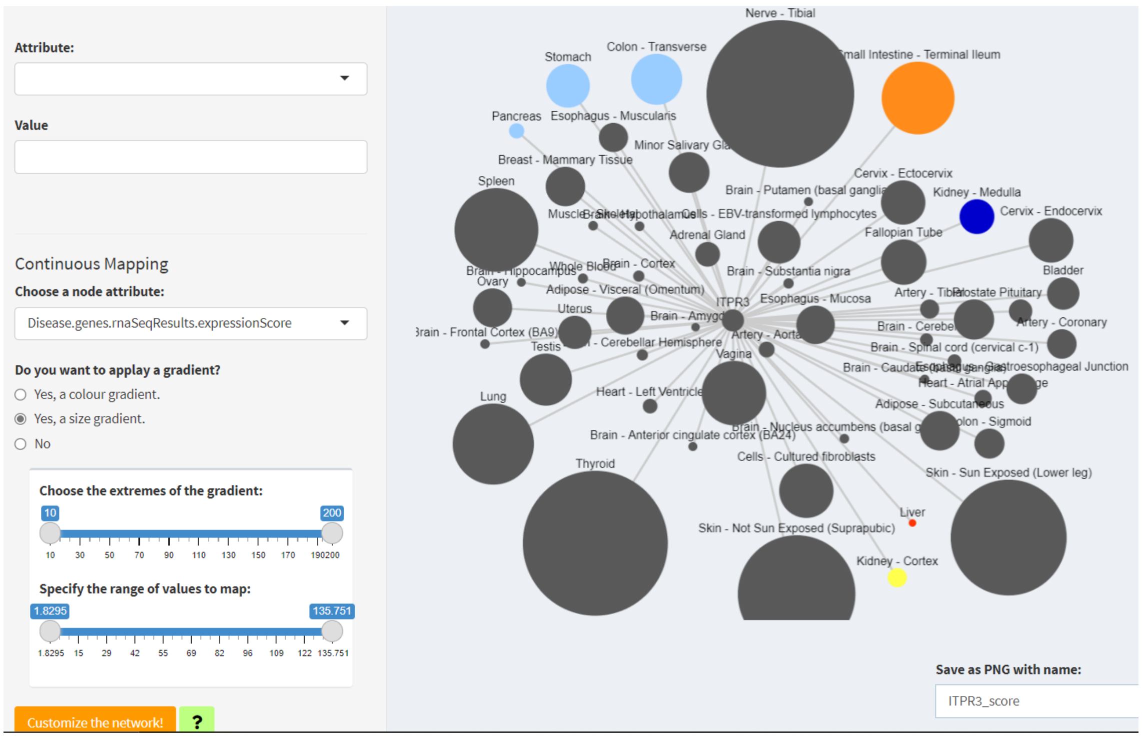
Summary

Show 10 entries Search:

	path	op	value	code
1	Disease.name	=	TYPE 1 DIABETES MELLITUS	A
2	Disease.genes.symbol	=	ITPR3	B
3	Disease.genes.rnaSeqResults.dataSets.name	=	RNA-Seq Data	C

Showing 1 to 3 of 3 entries

Figure C.33: Summary of constraints.



Attribute:

Value

Continuous Mapping

Choose a node attribute: Disease.genes.rnaSeqResults.expressionScore

Do you want to apply a gradient?

Yes, a colour gradient.

[✓ Yes, a size gradient.]

No

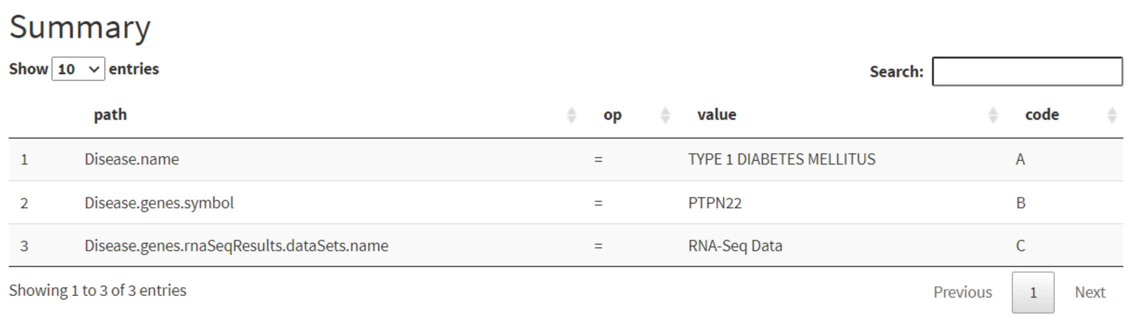
Choose the extremes of the gradient: 10 – 200

Specify the range of values to map: 1.8295 – 135.751

Save as PNG with name: ITPR3_score

Figure C.34: Size gradient of expression score for the ITPR3 gene.

C.1.2.4. Gene PTPN22:



Summary

Show 10 entries

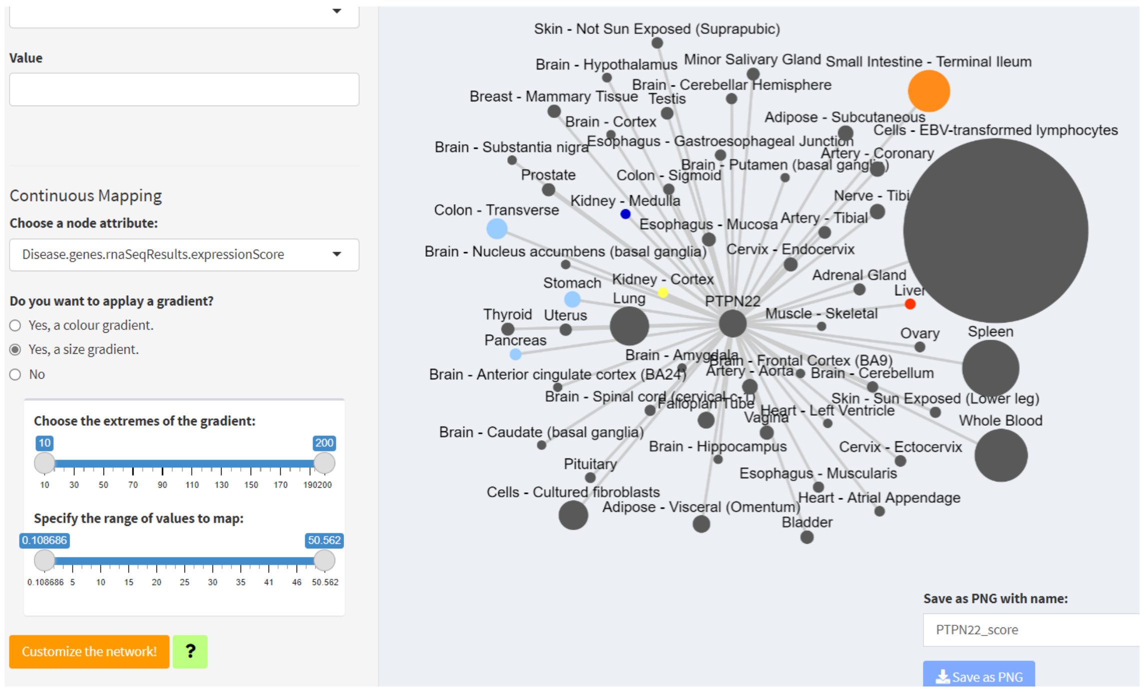
Search:

	path	op	value	code
1	Disease.name	=	TYPE 1 DIABETES MELLITUS	A
2	Disease.genes.symbol	=	PTPN22	B
3	Disease.genes.rnaSeqResults.dataSets.name	=	RNA-Seq Data	C

Showing 1 to 3 of 3 entries

Previous 1 Next

Figure C.35: Summary of constraints.



Value

Continuous Mapping

Choose a node attribute: Disease.genes.rnaSeqResults.expressionScore

Do you want to apply a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

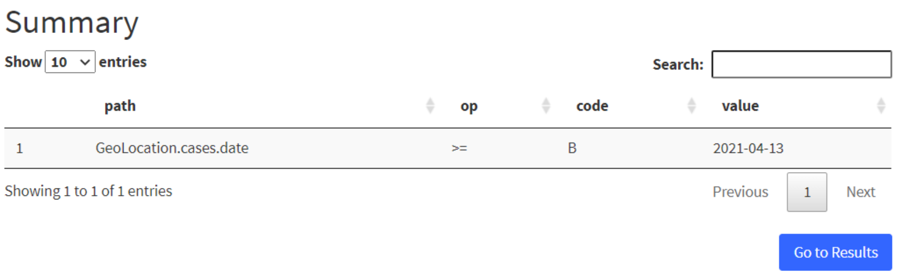
Choose the extremes of the gradient: 10 – 200

Specify the range of values to map: 0.108686 – 50.562

Save as PNG with name: PTPN22_score

Figure C.36: Size gradient of expression score for the PTPN22 gene.

C.2. CovidMine use-case.



Summary

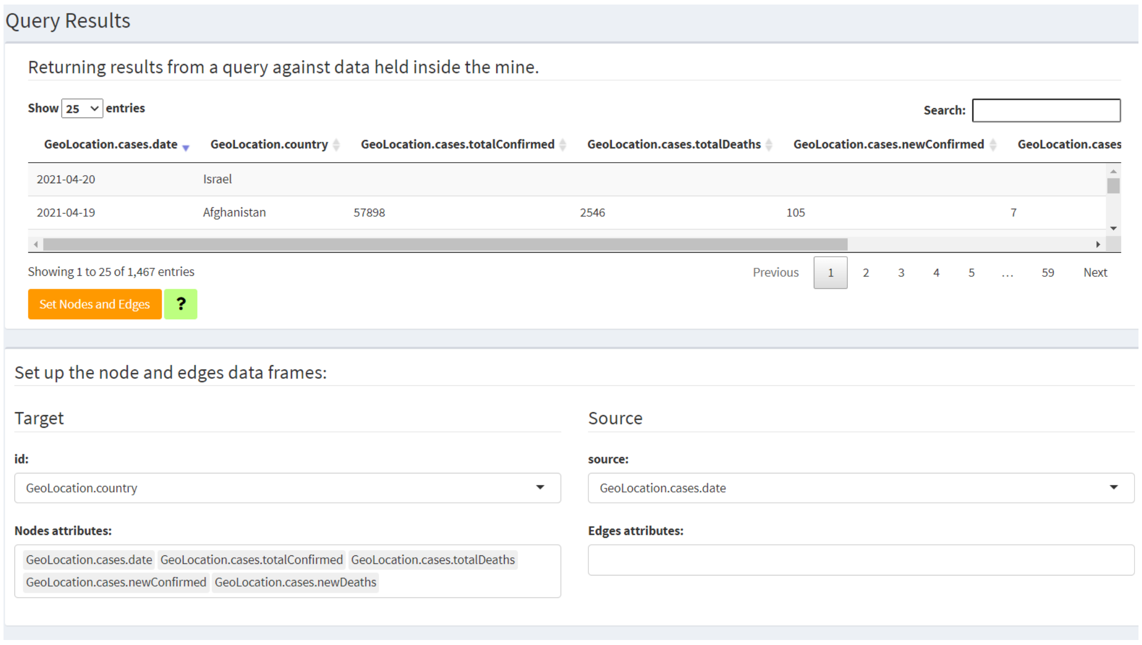
Show 10 entries

Search:

	path	op	code	value
1	GeoLocation.cases.date	>=	B	2021-04-13

Showing 1 to 1 of 1 entries

Figure C.37: Summary of constraints of the template query modified (new value for *date*).



Query Results

Returning results from a query against data held inside the mine.

Show 25 entries

Search:

GeoLocation.cases.date	GeoLocation.country	GeoLocation.cases.totalConfirmed	GeoLocation.cases.totalDeaths	GeoLocation.cases.newConfirmed	GeoLocation.cases
2021-04-20	Israel				
2021-04-19	Afghanistan	57898	2546	105	7

Showing 1 to 25 of 1,467 entries

Set up the node and edges data frames:

Target

id: GeoLocation.country

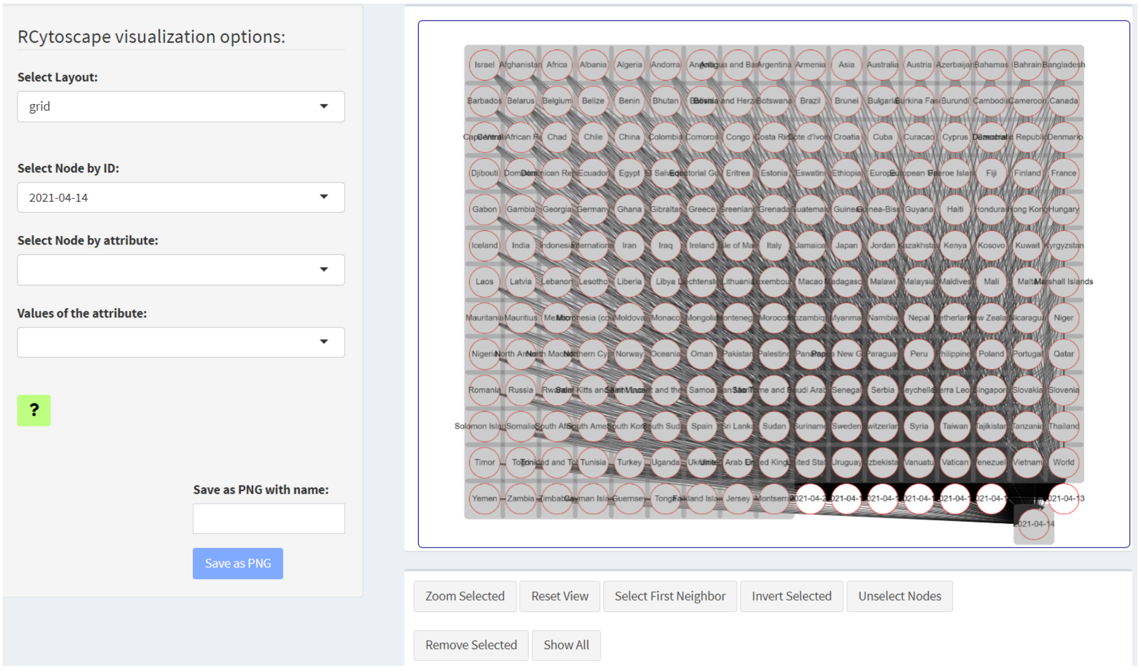
Nodes attributes: GeoLocation.cases.date GeoLocation.cases.totalConfirmed GeoLocation.cases.totalDeaths GeoLocation.cases.newConfirmed GeoLocation.cases.newDeaths

Source

source: GeoLocation.cases.date

Edges attributes:

Figure C.38: Query Results and selection of target and source data.


RCytoscape visualization options:

Select Layout: grid

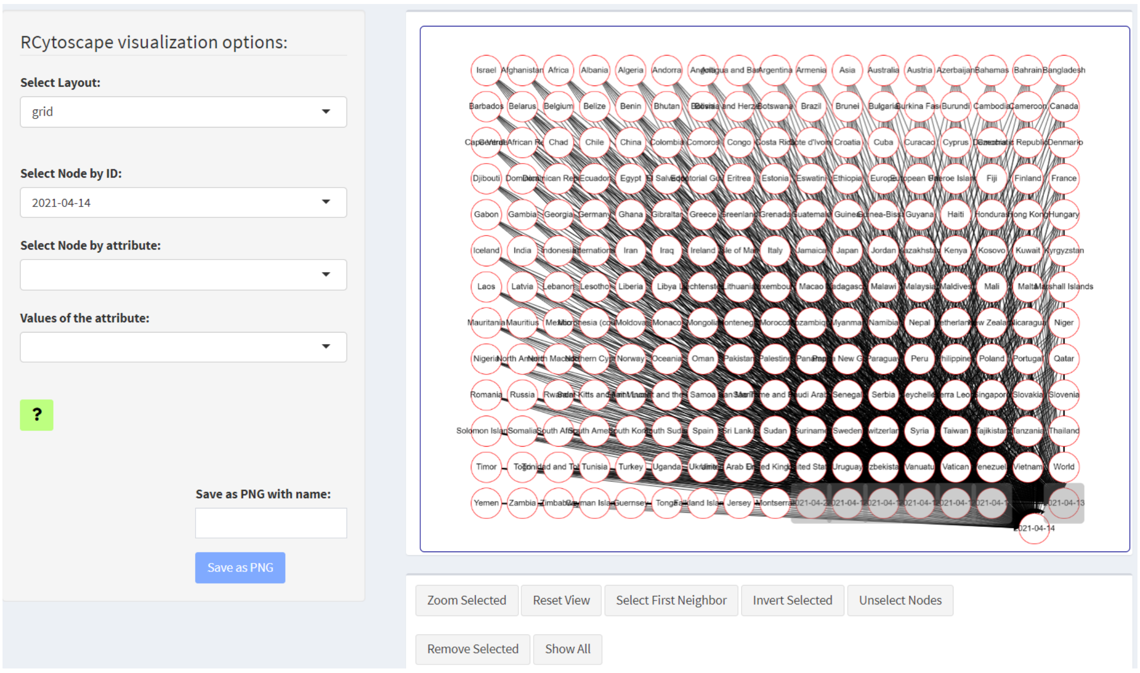
Select Node by ID: 2021-04-14

Select Node by attribute:

Values of the attribute:

Save as PNG with name:

Figure C.39: The node *2021-04-14* and its first neighbours are selected.


RCytoscape visualization options:

Select Layout: grid

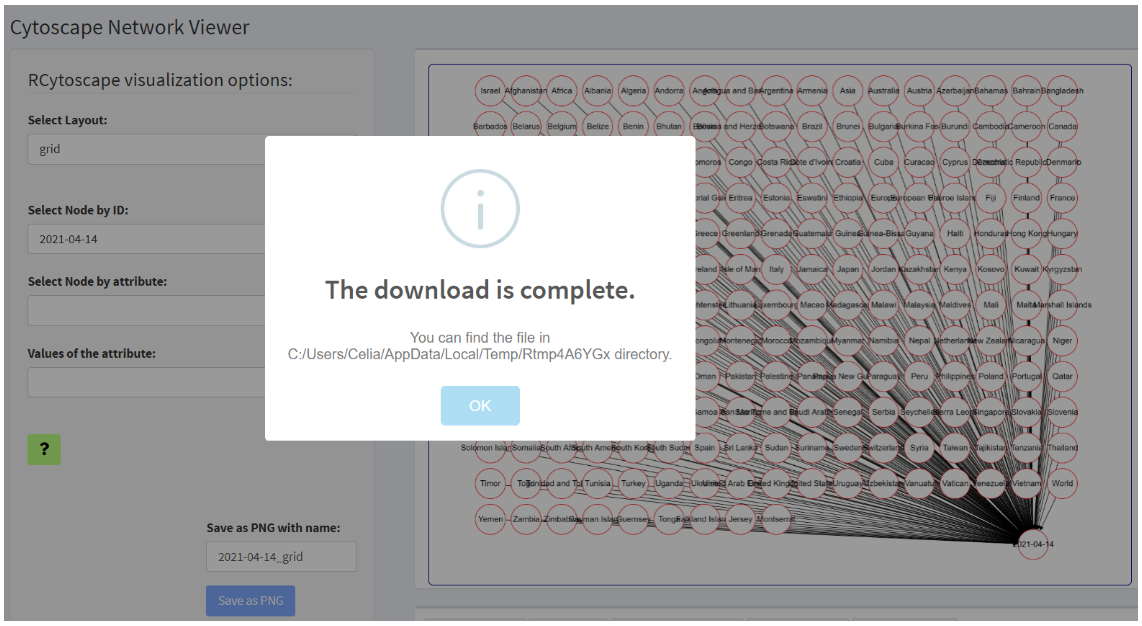
Select Node by ID: 2021-04-14

Select Node by attribute:

Values of the attribute:

Save as PNG with name:

Figure C.40: Inverted selection of C.39.


Cytoscape Network Viewer

RCytoscape visualization options:

Select Layout: grid

Select Node by ID: 2021-04-14

Select Node by attribute:

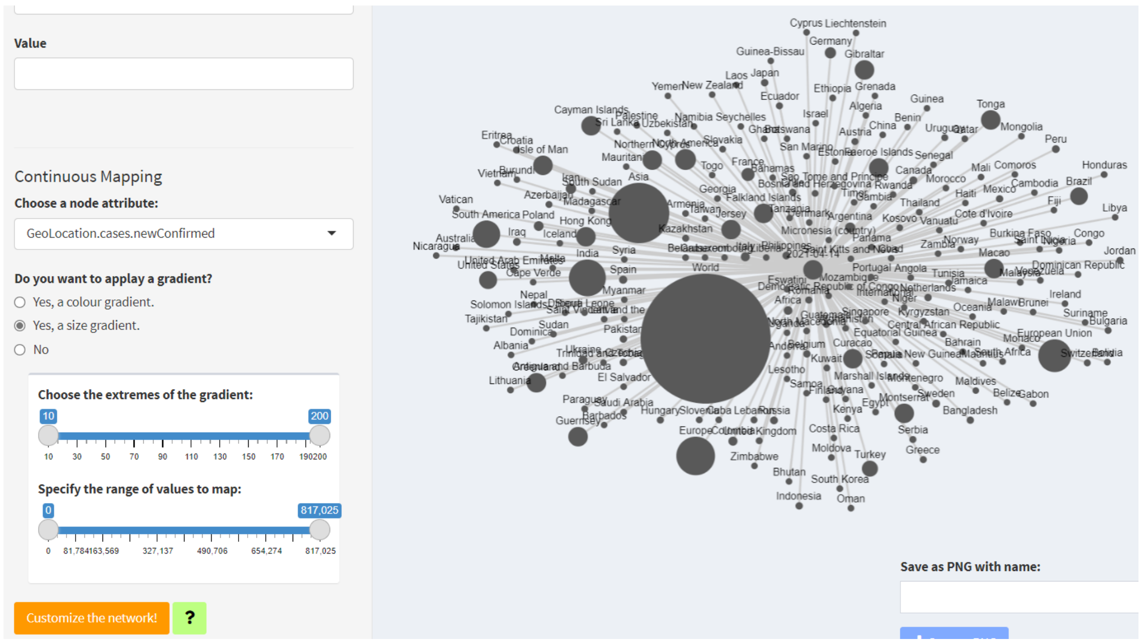
Values of the attribute:

The download is complete.

You can find the file in C:/Users/Celia/AppData/Local/Temp/Rtmp4A6YGx directory.

Save as PNG with name: 2021-04-14_grid

Figure C.41: Removing the nodes from C.40 selection and saving the results.


Value

Continuous Mapping

Choose a node attribute: GeoLocation.cases.newConfirmed

Do you want to apply a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

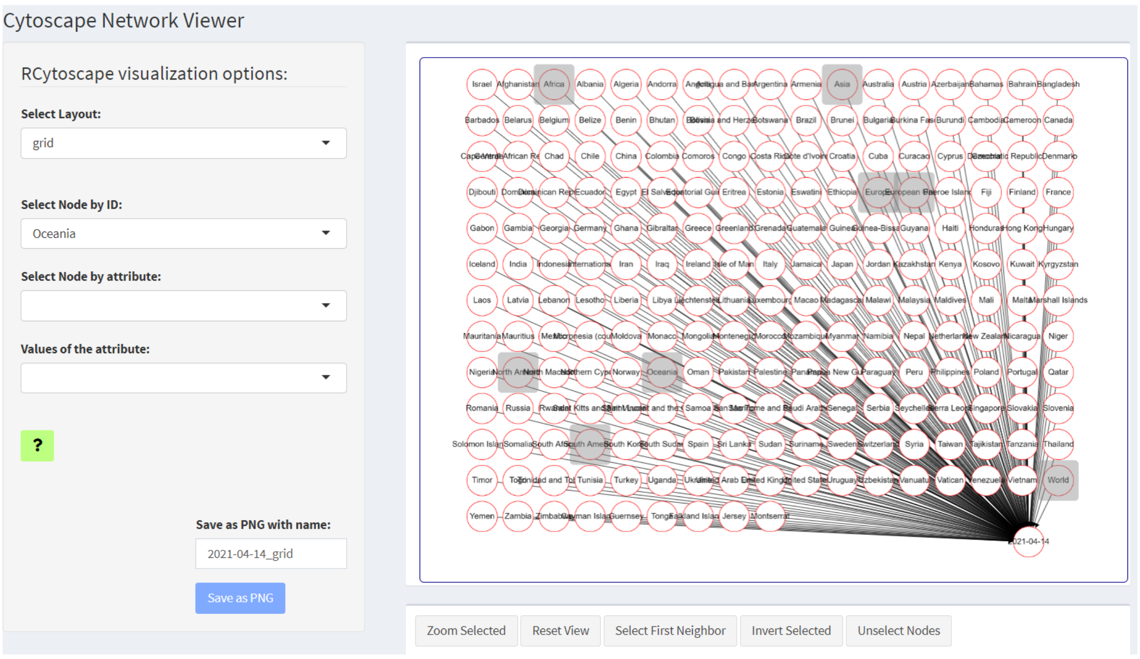
Choose the extremes of the gradient: 10 – 200

Specify the range of values to map: 0 – 817,025

Save as PNG with name:

Figure C.42: Size gradient of new confirmed Covid-19 cases on 14-04-2021.

C.2.1. Countries without continents nodes.


Cytoscape Network Viewer

RCytoscape visualization options:

Select Layout: grid

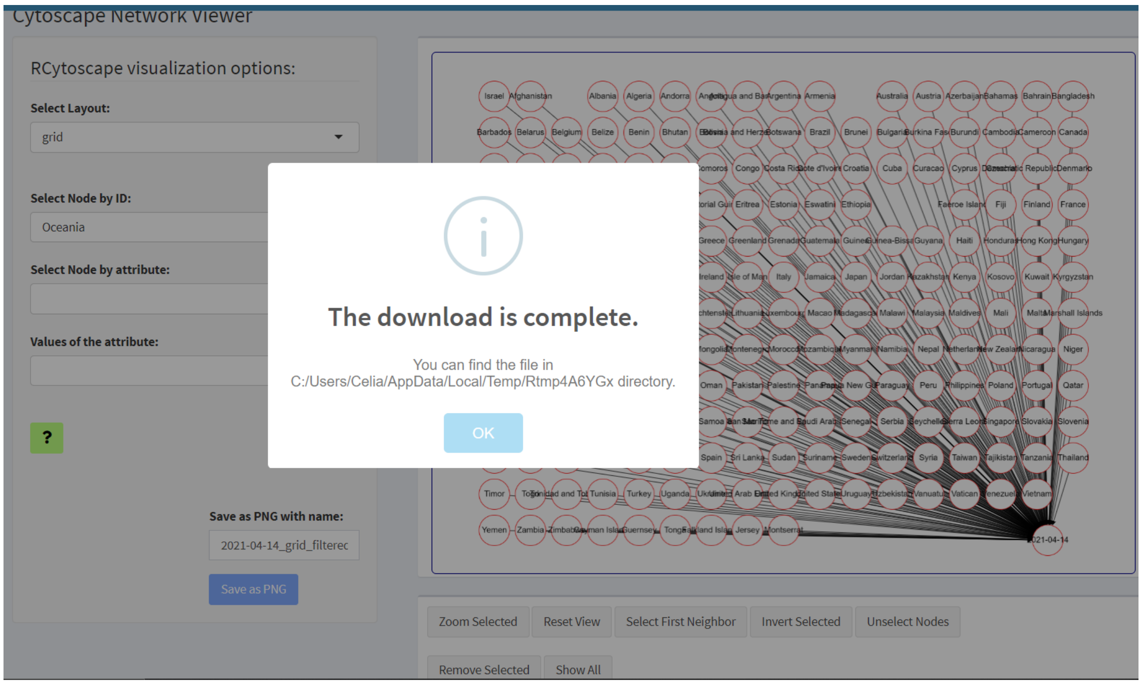
Select Node by ID: Oceania

Select Node by attribute:

Values of the attribute:

Save as PNG with name: 2021-04-14_grid

Figure C.43: Selection of continents.


Cytoscape Network Viewer

RCytoscape visualization options:

Select Layout: grid

Select Node by ID: Oceania

Select Node by attribute:

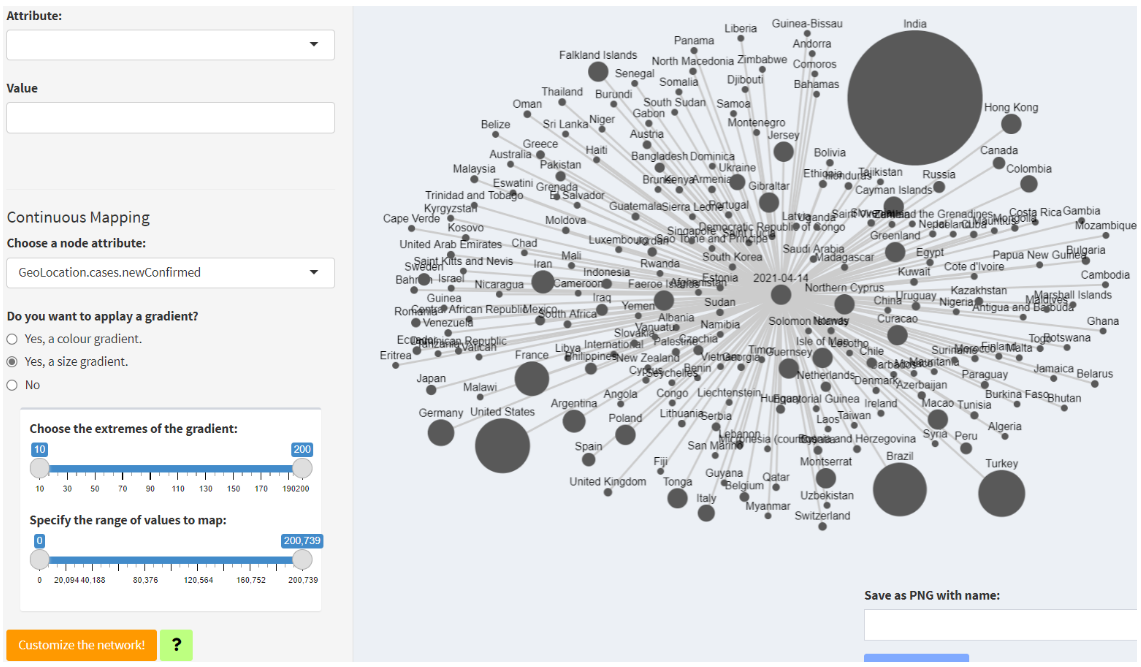
Values of the attribute:

Save as PNG with name: 2021-04-14_grid_filterec

The download is complete.

You can find the file in C:/Users/Celia/AppData/Local/Temp/Rtmp4A6YGx directory.

Figure C.44: Removing the nodes from C.43 and saving the results.



Attribute:

Value

Continuous Mapping

Choose a node attribute: GeoLocation.cases.newConfirmed

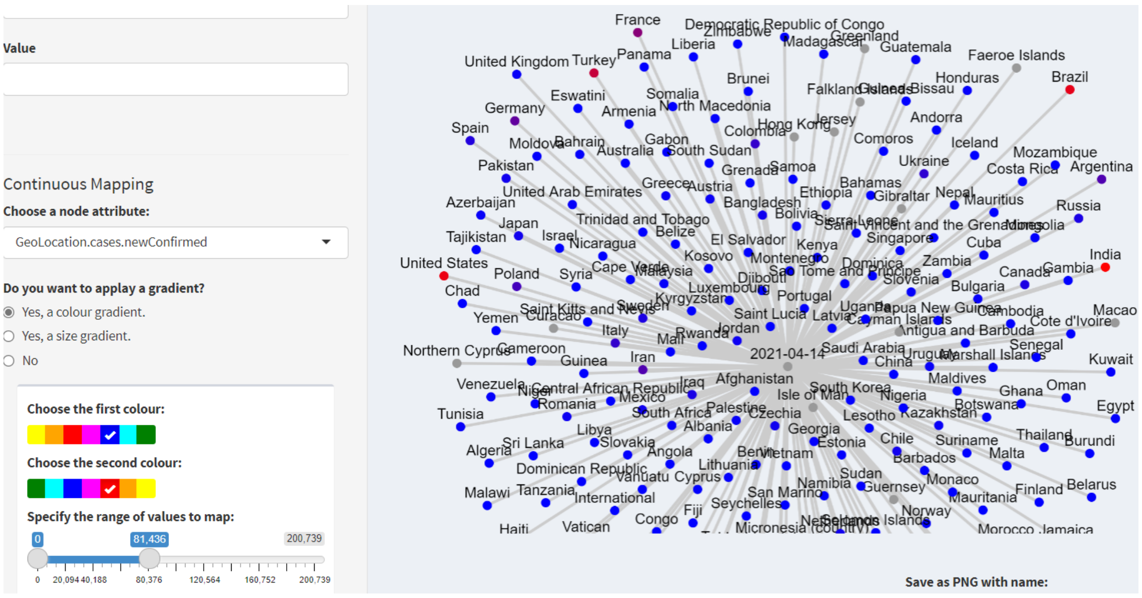
Do you want to apply a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

Choose the extremes of the gradient: 10 – 200

Specify the range of values to map: 0 – 200,739

Figure C.45: Size gradient of new confirmed Covid-19 cases on 14-04-2021 without continents.



Value

Continuous Mapping

Choose a node attribute:

GeoLocation.cases.newConfirmed

Do you want to apply a gradient?

[✓ Yes, a colour gradient.]
Yes, a size gradient.
No

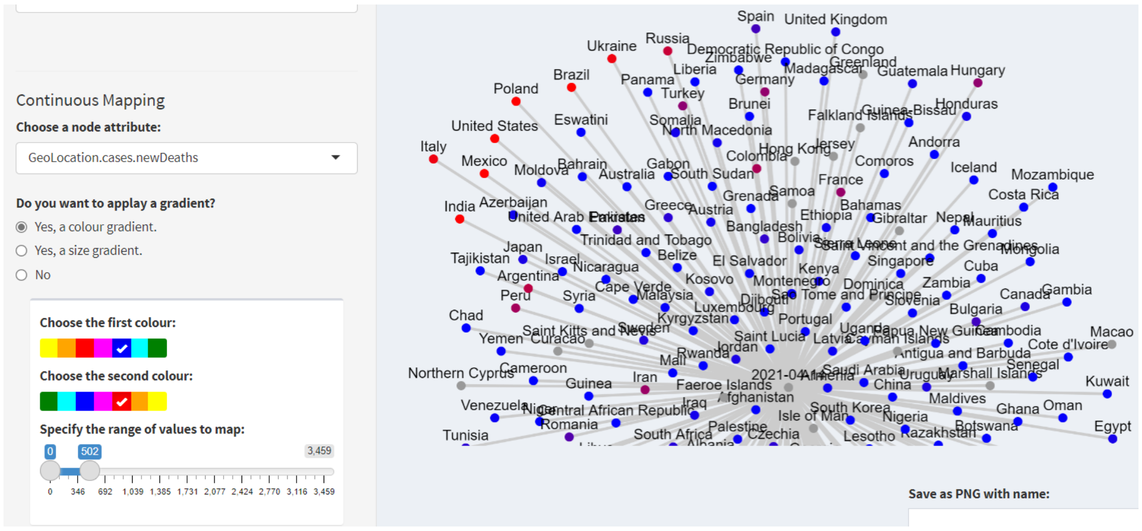
Choose the first colour:

Choose the second colour:

Specify the range of values to map:

0 81,436 200,739

(a) New confirmed Covid-19 cases.



Continuous Mapping

Choose a node attribute:

GeoLocation.cases.newDeaths

Do you want to apply a gradient?

[✓ Yes, a colour gradient.]
Yes, a size gradient.
No

Choose the first colour:

Choose the second colour:

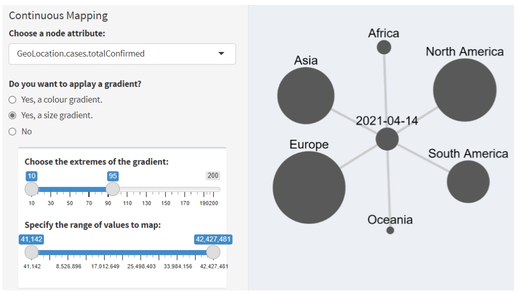
Specify the range of values to map:

0 502 3,459

(b) New deaths Covid-19 cases.

Figure C.46: Colour gradient (date: 14-04-2021).

C.2.2. Only continents.



Continuous Mapping

Choose a node attribute:

GeoLocation.cases.totalConfirmed

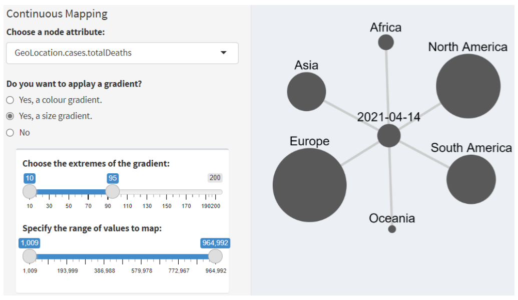
Do you want to applay a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

Choose the extremes of the gradient: 10 – 95 (scale 10 to 200)

Specify the range of values to map: 41,142 – 42,427,481

(a) Total confirmed Covid-19 cases.



Continuous Mapping

Choose a node attribute:

GeoLocation.cases.totalDeaths

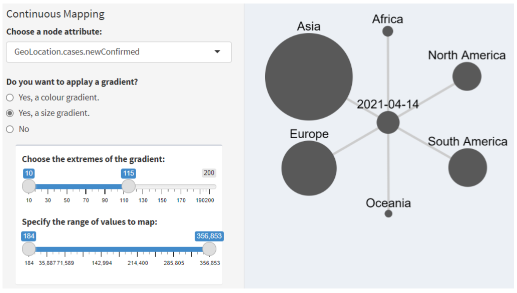
Do you want to applay a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

Choose the extremes of the gradient: 10 – 95 (scale 10 to 200)

Specify the range of values to map: 1,009 – 964,992

(b) Total deaths Covid-19 cases.



Continuous Mapping

Choose a node attribute:

GeoLocation.cases.newConfirmed

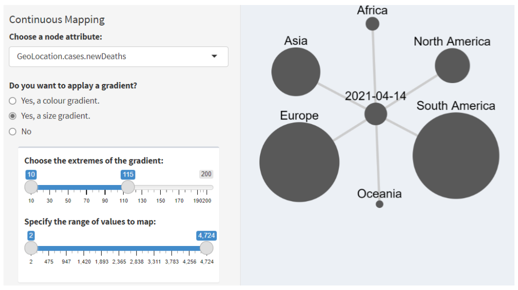
Do you want to applay a gradient?

Yes, a colour gradient. [✓ Yes, a size gradient.] No

Choose the extremes of the gradient: 10 – 115 (scale 10 to 200)

Specify the range of values to map: 184 – 356,853

(c) New confirmed Covid-19 cases.



Continuous Mapping

Choose a node attribute:

GeoLocation.cases.newDeaths

Do you want to applay a gradient?

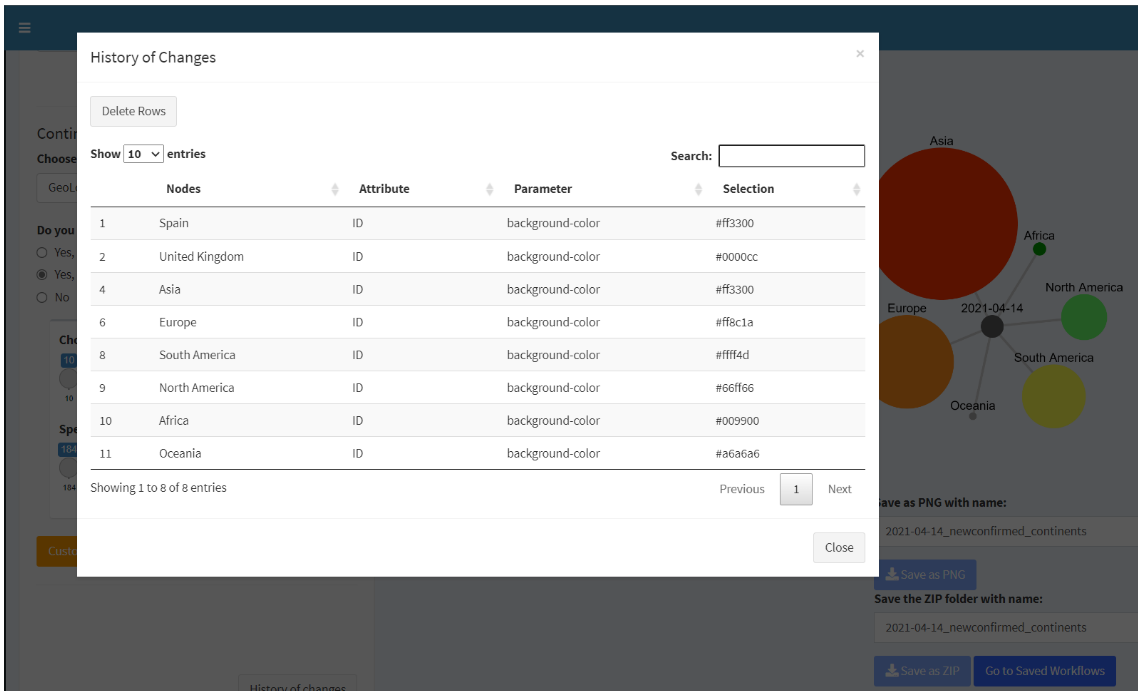
Yes, a colour gradient. [✓ Yes, a size gradient.] No

Choose the extremes of the gradient: 10 – 115 (scale 10 to 200)

Specify the range of values to map: 2 – 4,724

(d) New deaths Covid-19 cases.

Figure C.47: Size gradients (date: 14-04-2021).



History of Changes

Delete Rows

Show 10 entries Search:

	Nodes	Attribute	Parameter	Selection
1	Spain	ID	background-color	#ff3300
2	United Kingdom	ID	background-color	#0000cc
4	Asia	ID	background-color	#ff3300
6	Europe	ID	background-color	#ff8c1a
8	South America	ID	background-color	#ffff4d
9	North America	ID	background-color	#66ff66
10	Africa	ID	background-color	#009900
11	Oceania	ID	background-color	#a6a6a6

Showing 1 to 8 of 8 entries

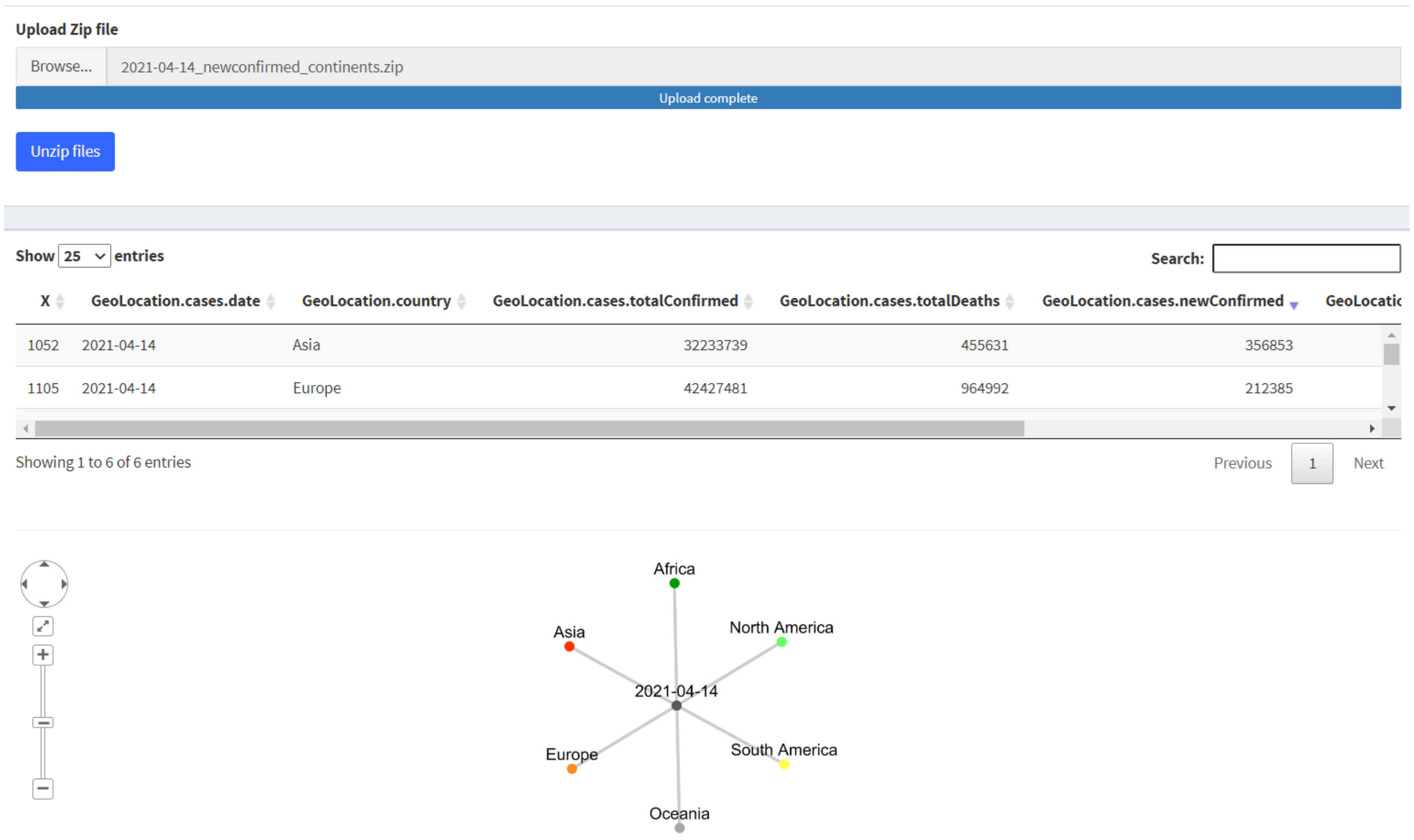
Previous 1 Next

Close

Save as PNG with name: 2021-04-14_newconfirmed_continents

Save the ZIP folder with name: 2021-04-14_newconfirmed_continents

Figure C.48: Size gradient of new confirmed Covid-19 cases on 14-04-2021 in the continents and background-colour overlaying seen in the History of Changes window.



Upload Zip file

Browse... 2021-04-14_newconfirmed_continents.zip

Upload complete

Unzip files

Show 25 entries

Search:

X	GeoLocation.cases.date	GeoLocation.country	GeoLocation.cases.totalConfirmed	GeoLocation.cases.totalDeaths	GeoLocation.cases.newConfirmed	GeoLocatic
1052	2021-04-14	Asia	32233739	455631	356853	
1105	2021-04-14	Europe	42427481	964992	212385	

Showing 1 to 6 of 6 entries

Previous 1 Next

Figure C.49: Displaying the network saved in C.48.

APPENDIX D. SUPPLEMENTARY GRAPHICS

D.1. Scan of the Market for Biological Data-Warehouses

		BioMart Ensembl	EuPathDB	BioCyc	InterMine
Databases		4	1,125	17,043	34
Content	Genomics	✓	✓	✓	✓
	Transcriptomics	✓	✓		✓
	Proteomics	✓	✓		✓
	Metabolomics		✓	✓	✓
Organism	Eukaryote	✓	✓	✓	✓
	Humans	✓		✓	✓
	Prokaryote	✓		✓	✓
	Viruses			✓	✓
Web Servers		Single	13	14	34
Personal Site			✓		✓
Query	Templates		✓	✓	✓
	Builder	By Filters and Attributes	✓	✓	✓Automatic Code Generation
	Superposition		Strategies can be created by adding, subtracting, joining, intersecting, or collocating the results of subsequent searches.	Advanced searches combining multiple organisms or types of objects.	
APIs		Pearl, Java		Java, Perl, Common Lisp languages	HTTP, Perl, Python, Ruby, JavaScript, R
R packages		✓	✓		✓
Visual Analysis Tools			✓	✓	✓

Table D.1: Comparison between BioMart, EuPathDB, BioCyc and Intermine.

D.2. Improvements of the core InterMineR package.

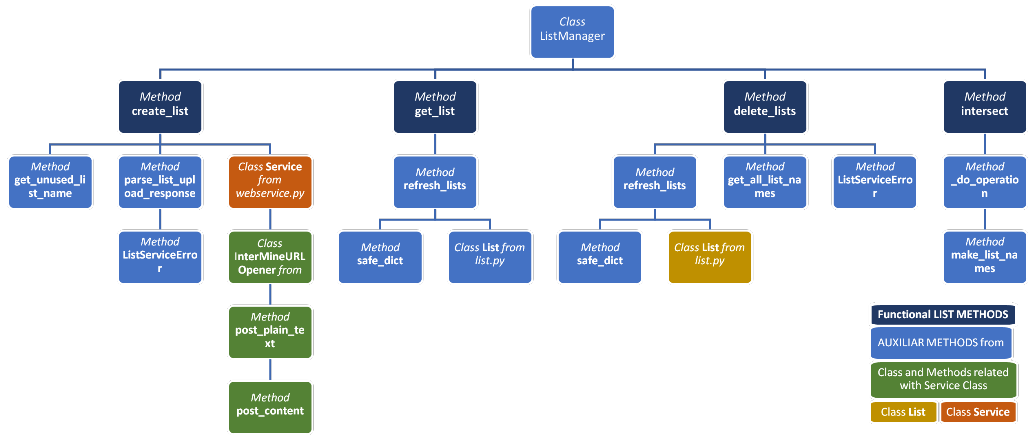
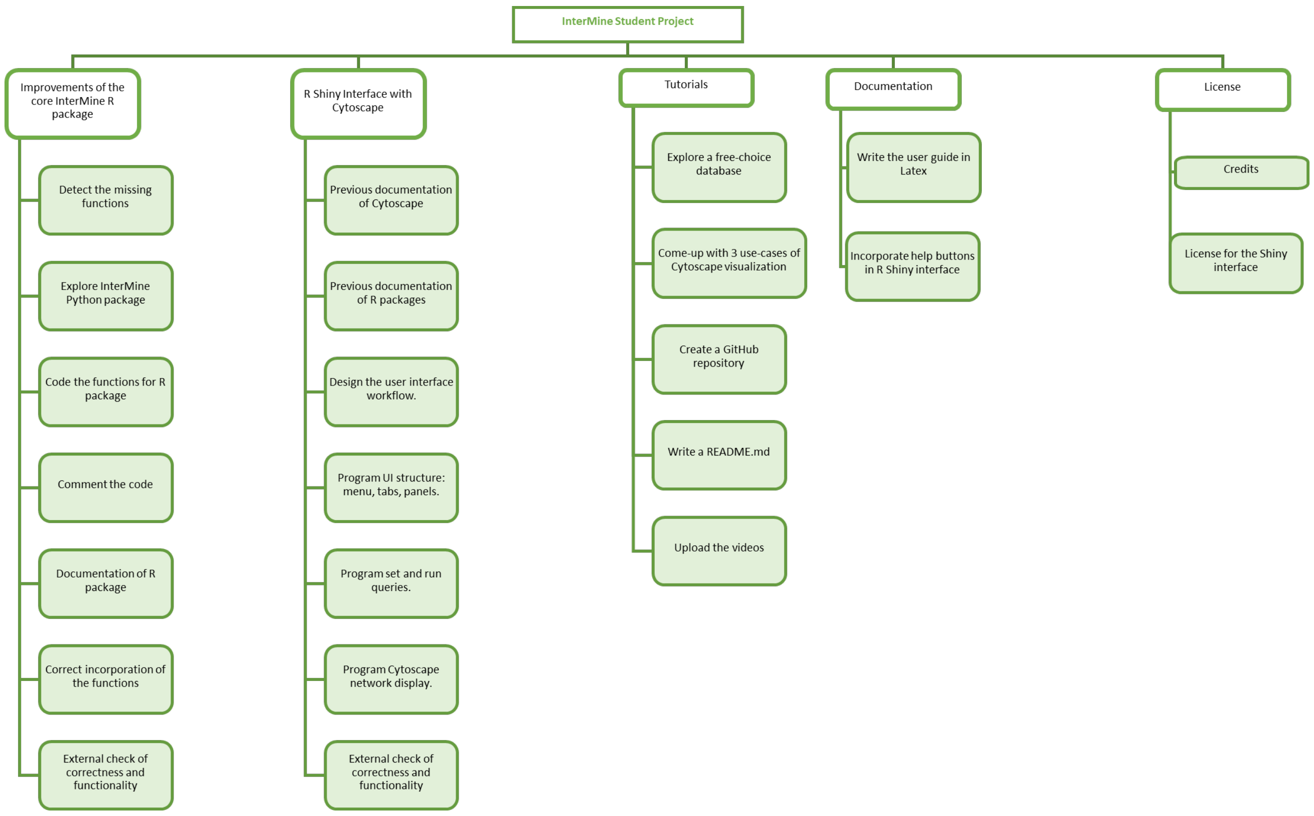


Figure D.1: Auxiliar scheme of the methods and classes to implement.

D.3. Work Breakdown Structure



InterMine Student Project
Improvements of the core InterMine R package
Detect the missing functions
Explore InterMine Python package
Code the functions for R package
Comment the code
Documentation of R package
Correct incorporation of the functions
External check of correctness and functionality
R Shiny Interface with Cytoscape
Previous documentation of Cytoscape
Previous documentation of R packages
Design the user interface workflow.
Program UI structure: menu, tabs, panels.
Program set and run queries.
Program Cytoscape network display.
External check of correctness and functionality
Tutorials
Explore a free-choice database
Come-up with 3 use-cases of Cytoscape visualization
Create a GitHub repository
Write a README.md
Upload the videos
Documentation
Write the user guide in Latex
Incorporate help buttons in R Shiny interface
License
Credits
License for the Shiny interface

Figure D.2: Work Breakdown Structure.