# The MetaboX Library: Building Metabolic Networks from KEGG Database

Francesco Maiorano[1], Luca Ambrosino[2] and Mario Rosario Guarracino[1]

[1] Laboratory for Genomics, Transcriptomics and Proteomics,
Institute for High-Performance Computing and Networking,
National Research Council of Italy
[2] Dept. of Agricultural Sciences, University of Naples Federico II

**Abstract.** In many key applications of metabolomics, such as toxicology or nutrigenomics, it is of interest to profile and detect changes in metabolic processes, usually represented in the form of pathways. As an alternative, a broader point of view would enable investigators to better understand the relations between entities that exist in different processes. Therefore, relating a possible perturbation to several known processes represents a new approach to this field of study. We propose to use a network representation of metabolism in terms of reactants, enzymes and metabolites. To model these systems, it is possible to describe both reactions and relations among enzymes and metabolites. In this way, analysis of the impact of changes in some metabolites or enzymes on different processes are easier to understand, detect and predict.

**Results.** We release the MetaboX library, an open source PHP framework for developing metabolic networks from a set of compounds. This library uses data stored in the *Kyoto Encyclopedia for Genes and Genomes* (KEGG) database using its RESTful *Application Programming Interfaces* (APIs), and methods to enhance manipulation of the information retrieved from the KEGG webservice. The MetaboX library includes methods to extract information about a resource of interest (e.g. metabolite, reaction and/or enzyme) and to build reactants network, bipartite enzyme-metabolite and unipartite enzyme networks. These networks can be exported in different formats for data visualization with standard tools. As a case study, the networks built from a subset of the Glycolysis pathway are described and discussed.

**Conclusions.** The advantages of using such a library imply the ability to model complex systems with few starting information represented by a collection of metabolites KEGG IDs.

## 1 Background

In metabolomics applications it is often of interest to model relationships and interactions among compounds and enzymes, such as protein-protein interactions, metabolite pathways or pathway flows. One of the main challenges in metabolomics is to model these interactions in the form of networks [1,2]. Such networks make it easier to understand the topological and functional structure of

F. Ortuño and I. Rojas (Eds.): IWBBIO 2015, Part I, LNCS 9043, pp. 565–576, 2015.

molecules and their interactions. Furthermore, once the network has been built, various statistics can be obtained for characterization and comparison. Indeed, a network represents a convenient way to model objects and their relationships as complex systems. Modeling a network of metabolites provides several ways to further analyze types of interactions, to understand the role of each metabolite in a particular pathway and to detect changes. The problem we address is to build *reaction*, *unipartite enzymes* and *bipartite enzyme-metabolite* networks, starting from a list of metabolites and information on metabolism gathered from a database. These networks models were used in other research studies in order to identify so-called reporter metabolites [3]. In a metabolite reaction network, two metabolites are connected if they are known to react in the same reaction. In a unipartite enzyme network, two enzymes are connected if they share at least one metabolite in the reactions they catalyze. In a bipartite enzyme-compound network, each enzyme is connected to every metabolite that is present in the reactions it catalyzes.

There are several publicly available databases that store and distribute information on molecular compounds providing different access methods. Among these we cite MetaCyc [4], EcoCyc [5], HMDB [6], Lipid Maps [7], BioCyc [8], Reactome [9], PubChem [10], Chebi [11], ChemSpider [12], Meltin [13], IIMDB [14], and KEGG [15]. It is out of the scope of this paper to describe the characteristics of all these databases, and we focus our attention on the latter.

KEGG is a database containing the largest collection of metabolites, as well as enzymes, reactions and other information[16]. It is possible to query the database with a web interface using one compound, and obtain information on the reactions in which it is involved, the stoichiometric equations, enzymes that catalyze the reaction, and metabolic pathways in which these reactions are involved. Its website graphically displays the stored pathways, but there is no functionality to build networks with a custom topology. To overcome these difficulties, some software exist, and partially solve these problems.

KEGGgraph [17] represents an interface between KEGG pathways and graph objects. It parses KGML (KEGG XML) files into graph models. This tool only provides modeling for KEGG pathways and it is only available for R. MetaboAnalyst [18] provides a web-based analytical pipeline for metabolomic studies. Its web interface can be used to load data as a list, for statistical analysis, as well as pathway analysis. When queried with a list of compounds, it returns information on pathways taken from KEGG but, for reasons related to XML representation of KEGG pathways, information concerning reactions and substrates are partially lost. Therefore, the resulting metabolic network is often disconnected and it does not represent a good model for graph analysis. The source code is not available, neither it provides APIs of any form.

INMEX [19], introduces an integrative meta-analysis of expression data and a web-based tool to support meta-analysis. It provides a web interface to perform complex operations step-by-step. While it supports custom data processing, annotation and visualization, it does not provide any APIs to extend core functionalities and it cannot be deployed in a custom environment. MetaboLyzer [20]

implements a workflow for statistical analysis of metabolomics data. It aims at both simplifying analysis for investigators who are new to metabolomics, and providing the flexibility to conduct sophisticated analysis to experienced investigators. It uses KEGG, HMDB, Lipid Maps, and BioCyc for putative ion identification. However, it is specifically suited for analysis of post-processed liquid chromatography-mass spectrometry (LCMS)-based metabolomic data sets. Finally, a tool that implements network construction is MetaboNetworks [21] which builds the networks using main reaction pairs and provide analyses for specific organisms. Although these software give the possibility to model a new network starting from a list of compounds, providing relevant tools for statistical and functional analyses, they miss the cabability to programmatically query metabolomics resources in order to develop novel applications and the software development choices made them suited for very specific environments raising difficulties to use them in production. With the aim to fill that gap, we introduce the MetaboX library which is a framework that enables investigators to extract information that is not visible at a first glance in KEGG. In fact, it is possible to retrieve many information available in the database with just a collection of KEGG IDs and then connect the gathered data in ways KEGG does not provide. The MetaboX library is written in PHP and it is platform indepentent. The latest version is available under the AGPL license on gitHub repository (https://github.com/Gregmayo/MetaboX-Library). On the other hand, it is possible to programmatically query the database, obtaining such information in the form of flat files. For large lists of input nodes it can be very difficult to manually gather information of interest to build a network, as well as other metadata useful to get a complete understanding of the biological system. With respect to the tools presented above, MetaboX provides a framework to model custom network layouts from a list of input nodes. Based on the nature of input nodes, we provide a set of classes to gather related information and programmatically build a network. The design of the MetaboX library is suited for web production environments, in fact it can be embedded in a custom webservice as it is released under the AGPL license. Therefore, the MetaboX library is an open source framework that aims to get a growing community of researchers and developers to support metabolomic analysis. With the MetaboX library, developers are able to model a network in different ways using the available methods to create a custom network layout that meets their needs. The library design is modular, with the aim to give developers the ability to implement different types of network builders from lists of compounds. Thus, gathering information and detecting interactions programmatically represents a benefit when working with large lists of metabolites. In the network construction process, MetaboX handles the following steps: *(i) connect* to the resource provider database using the PHP `libcurl` library. *(ii) query* a resource provider using methods to retrieve nodes and interactions. As KEGG does not provide a structured query response, we built a translation layer to extract information from flat files. *(iii) extract* requested resource attributes from returned data, parsing and storing them. This task is achieved using regular expressions. *(iv) cache* resource

attributes to file using a convenient data structure for serialization and for sharing and processing purposes such as JSON. *(v) build* a consistent data structure with information about requested resources using all previous steps, in order to build a network from collected data. Results consist of a weighted edgelist and a list of network nodes. It also includes specific resource information. In the *connect* step, MetaboX currently supports HTTP, HTTPS, FTP, and ldap protocols. It also supports HTTPS certificates, POST, PUT and FTP uploading, which are natively available in the *curl* library. It is also possible to *query* the KEGG database with a list of resources of interest and then parse the response to firstly separate information about each one and then extract specific data. In the *extract* step downloaded data are parsed to produce new files ready for next steps. We locally *cache* data to load resource information every time it is requested again by a new process. We design the caching system for MetaboX to speed up computation and to produce a sustainable amount of requests to the resource provider system. It is possible to invalidate the cache in order to reload updated data, and to manually delete the cache so that the library can update it. Finally, the *build* step is intended to put together all gathered information and output the resulting network. Every build method connects two nodes differently in each network model, that is *Reactant Graph* implementation of the build method is different from both *Enzyme Unipartite Graph* and *Enzyme Bipartite Graph*. In the following section, we report the implementation of the MetaboX library, detailing how it provides easy access to KEGG database and data manipulation. We explain how to use the library to build the different networks proposed and how to export the result for visualization and analysis with external tools like Cytoscape [22], which we use to render the figures presented in this paper. Then, we provide a case study and discuss the results. Finally, we conclude providing details on future work directions and open problems.

## 2 Implementation

At the moment of this writing, KEGG only offers a RESTful API interface thus MetaboX is designed to query these in an appropriate manner. KEGG used to expose SOAP APIs to standard software but these were suppressed on 31st december 2012 and the toolbox does not work anymore (https://www.biocatalogue.org/announcements/37). KEGG returns plain text upon web-service calls, thus making it necessary to parse results and arrange them in a data structure. We query KEGG multiple times and store the gathered information to file. The file format we use is JSON which is a lightweight data-interchange format, human-readable and writable. JSON is a text format that is completely language independent but uses conventions that are familiar to C-family programmers. These properties make JSON an ideal data-interchange language. To limit requests to KEGG, the MetaboX library loads previously processed resources from local storage, if they are available. A sample request for a resource in KEGG can be achieved using the following url: http://rest.kegg.jp/<operation>/<argument>. For instance, http://rest.kegg.jp/get/cpd:C01290 can be used to retrieve metabolite C01290.

**Network Construction.** We deal with compounds, reactions and pathways. To handle such a variety of entities, the MetaboX library defines proper classes. *AbstractResourceLoader* is an abstract class that provides methods needed to load an entity. To model an entity with a new class, this has to extend the abstract class and implement the abstract *load* method. When an entity is instatiated, this method first checks for existing records in the cache. If the requested entity has not been processed previously, a new file is built upon KEGG response. This pattern is used to load metabolites, reactions, pathways and enzymes. We provide several helper methods to extract information about resources from plain text using regular expressions. When the entity has been successfully processed, we serialize it to file for further reference. The attributes that define a metabolite are: *id*, *formula*, *exact mass*, *molecular weight*, *reaction list*, *pathway list*, *enzyme list*. Lists of other entities of interest that are related to a metabolite, such as reactions, pathways and enzymes, are loaded with different API call. For instance, if the *load* of C01290 returns a list of 10 reactions, we use a RESTful url to instantiate each of these reactions. It is possible to query KEGG RESTful APIs using collections of metabolites, reactions, enzymes or pathways. Using this capability, we designed the MetaboX library to construct queries splitting the input collection in chunks of 10 items (as this is the maximum chunk size KEGG supports). For reactions, we collect *id*, *name*, *definition*, *equation*, *enzymes* and *pathways*. For data manipulation purposes and to conveniently organize reaction information of input metabolites, we process reaction equations and split reactants from products in a data structure. Cache directories can be set in a configuration file. Each resource is stored in a dedicated resource directory and files are named after resource id (e.g. {resource}/{resource_id}.json would result in compound/C00002.json). The configuration file 'config.ini' is divided in sections and it is possible to specify storage directories for entities (e.g. config->directory->compound or config->directory->reaction) as well as KEGG API urls (e.g. config->url->compound or config->url->reaction). This approach is helpful if the entities become available in different urls or from another resource provider. In the MetaboX library we provide an interface to build several networks. *AbstractGraphBuilder* is an abstract class that defines the general structure of the resulting network. Specific network builder classes must implement the abstract *build* method provided in the abstract builder which takes one optional parameter. This is a list of metabolites out of which a sub network has to be built. To create a new type of network, a builder class should provide the construction of a network involving input metabolites and others involved in common reactions, or other entities, such as enzymes. If the optional parameter is specified, the builder method should create a network with set of nodes given by input parameter. When the network-construction process is completed, *getGlobalGraph* and *getSubGraph* methods return a multidimensional array containing the list of nodes, a weighted edgelist, where the weight represents the times a reaction has been found, and the list of connected and not connected nodes, in the case of a sub network.

**Reactants Network.** A network of reactants $G = (V, E)$ is an undirected graph where each node represents a metabolite and two given nodes $A$ and $B$ in $V$ interact with each other only if there is at least one reaction equation where $A$ and $B$ are involved as reactants. *ReactantsGraph* class builds a network out of a list of metabolites. To achieve this task, we first gather metabolites and reactions data from KEGG (Listing 1.1). We create a list of reactions that involve input metabolites and pass it to the class. In this case, the *build* method cycles through the list of reactions and, for each one, the list of substrates is extracted. We then connect each substrate to one another and when all direct network interactions have been built, we produce a weighted edgelist. Such edgelist represents a network including input compounds and all other compounds involved in processed reactions. We also save a weighted list of interactions that only include input compounds, this resulting in a smaller network which can be seen as a sub network of the global weighted interaction list. As shown in Fig. 1, the sub network is embedded in the global one. A builder class exposes methods to compute results and pass them to the graph writer classes in order to produce a file format that is suitable to the needs of further analysis, such as SIF and XML. The modeling of this class of networks allows to detect which compounds are directly connected, being reagents of the same reactions. It highlights what are the highly connected hubs in a network made up of the collection of metabolites under analysis. This information is useful for planning metabolic engineering strategies. It is clear that if we wish to modify a node of this type of network, it is crucial to know what are other reactants to be considered, so that the change can effectively impact on the metabolic system of the studied biological organism.

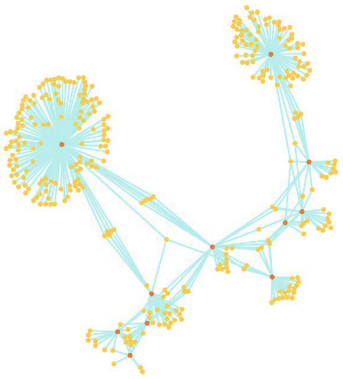
```
// Retrieve and collect compound information
foreach( $compounds as $compound ){ $_cpd_id = trim($compound);
    $cpd_loader = new MetaboX\Resource\Loader\Compound($_cpd_id, $cpdLoaderConfig);
    $_compounds[$_cpd_id] = $cpd_loader->load();
}

// Retrieve and collect reactions information
foreach($_compounds as $id => $compound){
    $rn_list = $compound->reactionIdCollection;

    if( $rn_list ){
        foreach( $rn_list as $rn ){ $_rn_id = trim($rn);
            $rn_loader = new MetaboX\Resource\Loader\Reaction($_rn_id, $rnLoaderConfig);
            $_reactions[$_rn_id] = $rn_loader->load();
    }   }   }

// Create reactants graph
$_graph = new MetaboX\Graph\ReactantsGraph($_reactions);
$_graph->build($compounds);
```

**Listing 1.1.** Loading Metabolites and Reactions metadata from KEGG

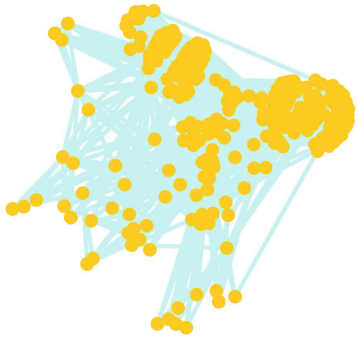
```
// Retrieve and collect reactions information
foreach($_compounds as $id => $compound){
    $ec_list = $compound->enzymeIdCollection;

    if( $ec_list ){
        foreach( $ec_list as $ec ){
            $_ec_id = trim($ec);
            $ec_loader = new MetaboX\Resource\Loader\Enzyme($_ec_id, $ecLoaderConfig);
            $_enzymes[$_ec_id] = $ec_loader->load();
        }
    }
}
```

**Listing 1.2.** Loading Enzymes metadata from KEGG

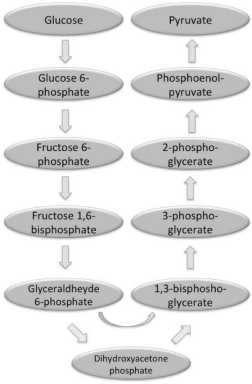
(1) A network of reactants obtained from the 11 input metabolites (darker nodes) selected from glycolysis pathway. This network shows 108 nodes and 151 edges.



(2) Enzyme-metabolite bipartite network: 342 nodes and 393 interactions. Darker nodes represent metabolites.



(3) Enzymes unipartite network: 297 nodes and 7705 interactions.




(4) A standard view of glycolysis.

**Bipartite Enzyme-Metabolite Network.** A network of enzymes and metabolites is a bipartite undirected graph $Z = (U, V, E)$ with set of nodes $U$ representing metabolites and $V$ representing enzymes. A metabolite node is connected to all the enzymes nodes that catalyze a reaction involving that metabolite, and an enzyme node is connected to all the metabolites that take part in the corresponding reaction. That is, if an enzyme $F$ in $V$ catalyzes a reaction where a metabolite $M$ in $U$ is a substrate, then an interaction between $F$ and $M$ exists in the network. We achieve this task using *EnzymeBipartiteGraph* class which

parameters are: a metabolite collection, an enzyme collection and a reaction collection. We cycle through the list of metabolites and select the related enzymes. We search current metabolite $M$ in the substrates of the reaction catalyzed by enzyme $F$. If we have a match, we connect nodes $F$ and $M$. An enzymes network, both unipartite and bipartite, provides a kind of visualization that highlights some aspects that are not observable by a reactants network. If we are analyzing different time conditions with different concentration levels of some compounds, for instance, this class of networks would quickly identify which nodes are most affected, restricting the area of interest to the enzyme directly susceptible to a particular condition. Therefore, the construction of this type of graphs can help highlight changes in the enzymatic expression levels or to detect enzymes with structural or functional defects due to particular conditions of stress. An example of such a network is shown in Fig. 2.

**Unipartite Enzymes Network.** A unipartite network of enzymes is an undirected graph $G = (V, E)$ where nodes represent enzymes and two enzymes sharing a common compound in the corresponding reactions are connected to each other. The class used to model such a network is *EnzymeUnipartiteGraph*. This builder class is instantiated with a list of enzymes and a list of reactions. These lists are created collecting all reactions and enzymes that involve input metabolites (Listing 1.2). For each enzyme in the collection, we load data of the reaction catalyzed and select all substrates. We cycle through the enzyme collection comparing the current enzyme substrates to all others. Given two enzymes $T$ and $S$ in $V$, we connect them if the intersection between substrates in $T$ and substrates in $S$ is not empty. An example of such a network is shown in Fig. 3.

**Data Export.** In the MetaboX library, there are two classes that can be used to export the constructed network in other formats. As for the other components, a *AbstractGraphWriter* is an abstract class that exposes an abstract *write* method. The class constructor takes one parameter, that is a multidimensional array containing the node list and the weighted edgelist of the network. This can be set using *getGlobalGraph* or *getSubGraph* to export respectively a network or a sub network. The *write* method has two parameters: the name of the file to be written and the data that needs to be exported. If the output needs to be prepared or modified somehow, it is possible to call *prepareOutput* within the *write* method. This is the case of *CytoscapeGraphWriter* class where interactions are converted to string and then written to file. To work with the D3JS visualization library as well as D3py or NetworkX [25], the MetaboX library provides several classes to export a network in one of the formats accepted by other analysis tools. For instance, a *D3JSGraphWriter* converts the network to JSON and writes it to file.

## 3 Results and Discussion

In order to test MetaboX, we build a network starting from a set of eleven compounds, listed below: Glucose (C00031), Glucose 6-phosphate (C00668), Fructose

6-phosphate (C05345), Fructose 1,6-bisphosphate (C05378), Dihydroxyacetone phosphate (C00111), Glyceraldehyde 3-phosphate (C00118), 1,3-bisphosphoglycerate (C00236), 3-phosphoglycerate (C00197), 2-phosphoglycerate (C00631), Phosphoenolpyruvate (C00074), Pyruvate (C00022). These compounds belong to glycolysis, the metabolic pathway that leads, starting from glucose, to the production of pyruvic acid through several reactions. Fig. 1 shows the network of all reactions involving input metabolites. This results in a network that includes input metabolites as well as others related to them. Here we find 151 interactions between 108 metabolites, including all the input metabolites. An interaction in this graph means to be reactants of the same reactions, therefore each node is directly affected by a decrease in the concentration of one of its neighbors, not by an increase. In this second case, in fact, there would be an excess of one of the two reactants. For instance, this information is useful if we want to plan a change in the levels of a compound starting from other compounds already known. Glucose and Pyruvic acid, as well as representing the start and the end of the glycolysis, are also hubs, namely highly connected nodes, in the network shown in Fig. 1.The MetaboX library also outputs a reactants subnetwork which only contains the nodes and the edges of the input metabolites. In this case study, the subnetwork results in just few nodes and edges (3 nodes and 2 edges) because the compounds choosen from glycolysis represent a subset of the glycolysis pathway (map00010) shown in Fig. 4. Therefore, each metabolite is both a reactant and a product of its neighbors in the pathway. The networks built with the MetaboX library are not pathway representation of the input metabolites. A classic pathway view in the form of substrate-product flow is provided by many databases of metabolic data (KEGG, MetaCyc, MetaboAnalyst), and it is not implemented in the MetaboX library. Instead, we mainly focus our efforts to highlight other information, like the relationships between metabolites of the same reaction. In this case study, compounds do not follow the common representation of glycolysis. Indeed, they do not show the same connections in the reactants network built with the MetaboX library. Another example we provide is the construction of a unipartite and bipartite enzymes network. In the first case, in Fig. 3, each node of the network is an enzyme, and two of them are connected if they share at least one metabolite in the reactions they catalyze, with the constraint that the substrate of one enzyme is the product of another enzyme. We add this constraint to allow the user to easily have a view of the substrate-product flow, looking at enzymes instead of metabolites. In this way, we are able to build a unipartite enzymes network with 297 nodes and 7705 interactions. Finally, we build a bipartite enzyme-metabolite network, in Fig. 2. As already mentioned, this network consists of two sets of nodes: metabolites and enzymes. Nodes are connected alternatively, that is a metabolite to an enzyme or vice versa. Connections between two metabolites or two enzymes are not possible. In the resulting network, we are able to find 342 nodes (11 metabolites and 331 enzymes) and 393 interactions. Looking at this network, we can easily identify the hubs (namely glucose, pyruvate, glyceraldehyde 3-phosphate and phosphoenolpyruvate) and all the enzymes related

to them. To change something in these hubs, the variables (enzymes) to take into account can be many, and the design of a subsequent experiment in metabolic engineering would be too complicated. A better solution is represented by focusing on compounds that show only few connections, in order to limit the analysis to few enzymes and to decrease the complexity of further analyses. Anyhow, we strongly believe that this type of network significantly simplifies the work of those who analyze metabolic pathways to understand metabolic disorders, to connect disease to enzyme defects, to design successful metabolic engineering strategies. MetaboNetworks provides analyses for specific organisms whereas in the MetaboX library we do not account for this feature, considering all available reactions in the first place. In such a way, a user can plan pipelines or methodologies from a compound-wise point of view, and not an organism-wise point of view. For instance, in soil remediation from copper, MetaboX provides access to all known reactions containing copper. A user can then find which organisms use copper within their metabolic pathways. In the enzyme API call, KEGG provides the *GENES* attribute containing a list of genes where that particular enzyme is involved. Each one of those genes is specific for an organism, enabling the user to filter organism specific reactions. In conclusion, if we start without any initial information about compounds concentration and we look at the topology of the network, the highly connected nodes are fundamental in the metabolism, and changes in these nodes would have probably led to the death of the organism. On the contrary, if we start from experimental data, it might be useful to correlate increases or decreases of the concentration of some compounds to a particular disease or to a particular disorder. Therefore highlighting compounds within a network should be useful in designing any strategy aimed at clarifying ways of occurrence of the disease, extracting from that network information like mumber of edges, enzymes, reactions, etc. The MetaboX library is a suitable tool created to solve both issues: a first preliminary view and a second in-depth analysis.

## 4 Conclusions

In this paper we describe the MetaboX library, a framework to build metabolic networks using information gathered from the KEGG database. The advantages of using such a library are: *(i)* the possibility to gather information from KEGG using a collection of KEGG IDs. *(ii)* the possibility to build a representation of the metabolic processes that can highlight how changes in metabolites or enzymes might affect other processes. *(iii)* the possibility to export the networks to other formats for visualization and analysis with standard software, such as Cytoscape, NetworkX, D3JS or D3py. Because of its extensibility, the MetaboX library may add support to other fields as in the construction of protein-protein interaction (PPI) networks for performing different topological and functional analyses [23]. In this case, the MetaboX library should use a resource provider that stores information about interactions between proteins such as STRING [24]. An organism filter can be implemented in the MetaboX library, as used

in MetaboNetworks, in order to select specific reactions and build a sub network that enables an organism-wise network. This can be achieved building an organism list for each enzyme of each processed reaction.

The library has been built with the possibility to extend data information gathering, such as downloading these from databases other than KEGG or merge information collected from multiple databases. Indeed, in order to make the network construction as complete as possible, the MetaboX library will implement a merge process among different resource providers. As KEGG information is limited, it makes sense to gather data about metabolites, reactions, pathways and enzymes from other databases like MetaCyc.

The MetaboX library is the starting point of a three layer project involving a web service and a web application. "Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.". Following this vision, MetaboX library offers a framework to work with metabolic networks. On top of that, we will develop a web service to expose core functionalities to the web. The MetaboX webservice will work in a RESTful fashion, providing APIs to retrieve resources information, network construction options and job submission. Moreover, we will implement alternative ways to make data persistent, such as database storage.

**Acknowledgements.** This work has been partially funded by MIUR projects PON02_00619 and Italian Flagship project *Interomics*. We wish to thank LabGTP (Laboratory for Genomics, Transcriptomics and Proteomics) researchers from ICAR CNR for helpful discussion during the early stages of this project and for testing the MetaboX library giving an important feedback for improvements.

## References

1. H.K., et al.: Metabolic network modeling and simulation for drug targeting and discovery. Biotechnol J., 30–42 (2011)
2. Cloots, L., et al.: Network-based functional modeling of genomics, transcriptomics and metabolism in bacteria. Curr Opin Microbiol, 599–607 (2011)
3. Raosaheb, K., et al.: Uncovering transcriptional regulation of metabolism by using metabolic network topology. PNAS, 2685–2689 (2005)
4. Krieger, C.J., et al.: MetaCyc: a multiorganism database of metabolic pathways and enzymes. Oxford Journals Nucl. Acids Res, 511–516 (2004)
5. Keseler, I.M., et al.: EcoCyc: fusing model organism databases with systems biology, Oxford Journals Nucl. Oxford Journals Nucl. Acids Res, D605-D612 (2013)
6. Wishart, D.S., et al.: HMDB: the Human Metabolome Database. Nucleic Acids Res, D521-D526 (2007)
7. Sud, M., et al.: LMSD: LIPID MAPS structure database. Oxford Journals Nucl. Acids Res, 527–532 (2007)
8. Karp, P.D., et al.: Expansion of the BioCyc collection of pathway/genome databases to 160 genomes. Nucl. Acids Res, 6083–6089 (2005)
9. Matthews, L., et al.: Reactome knowledgebase of human biological pathways and processes. Nucl. Acids Res, D619-D622 (2009)

10. Wang, Y., et al.: PubChem's BioAssay Database, Nucl. Nucl. Acids Res, D400-D412 (2012)
11. Hastings, J., et al.: The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013. Oxford Journals Nucl. Acids Res, 456–463 (2013)
12. Pence, H.E., et al.: ChemSpider: An Online Chemical Information Resource. J. Chem. Educ., 1123–1124 (2010)
13. Smith, C.A., et al.: METLIN: A Metabolite Mass Spectral Database Therapeutic Drug Monitoring. In: Proc. of the 9th ICTDM, pp. 747–751 (2005)
14. Menikarachchi, L.C., et al.: Silico Enzymatic Synthesis of a 400.000 Compound Biochemical Database for Nontargeted Metabolomics. J. Chem. Inf. Model., 2483–2492 (2013)
15. Kanehisa, M., et al.: KEGG for integration and interpretation of large-scale molecular data sets. Nucl. Acid Res. 14, D109-D114 (2011)
16. Altman, T., et al.: A systematic comparison of the MetaCyc and KEGG pathway databases. BMC Bioinformatics (2013)
17. Jitao, D.Z., et al.: KEGGgraph: a graph approach to KEGG PATHWAY in R and bioconductor. Bioinformatics, 1470–1471 (2009)
18. Xia, J., et al.: MetaboAnalyst 2.0 - a comprehensive server for metabolomic data analysis. Nucl. Acids Res., 1–7 (2012)
19. Xia, J., et al.: INMEXa web-based tool for integrative meta-analysis of expression data. Nucl. Acids Res., W63-70 (2013)
20. Mak, T.D., et al.: MetaboLyzer: A Novel Statistical Workflow for Analyzing Post-processed LCMS Metabolomics Data. Anal. Chem. Article ASAP, 506–513 (2013)
21. Posma, J.M., et al.: MetaboNetworks, an interactive Matlab-based toolbox for creating, customizing and exploring sub-networks from KEGG. Bioinformatics (2013)
22. Cline, M.S., et al.: Integration of biological networks and gene expression data using Cytoscape. Nat Protoc., 2366–2382 (2007)
23. Sharma, A., et al.: Rigidity and flexibility in protein-protein interaction networks: a case study on neuromuscular disorders, arXiv, arXiv:1402.2304v2 (2014)
24. Franceschini, A., Szklarczyk, D., et al.: STRING v9.1: protein-protein interaction networks, with increased coverage and integration. Nucleic Acids Res 41, D808–D815 (2013)
25. Hagberg, A.A., et al.: Exploring Network Structure, Dynamics, and Function using NetworkX. Proc. SciPy, 11-16 (2008)