Data integration for biological network databases: MetNetDB labeled graph model and graph matching algorithm

Jie Li
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/etd

Part of the Cell and Developmental Biology Commons, and the Genetics and Genomics Commons

Recommended Citation

This Dissertation is brought to you for free and open access by the Graduate College at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Data integration for biological network databases: MetNetDB labeled graph model and
graph matching algorithm

by

Jie Li

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-majors: Bioinformatics and Computation Biology; Computer Science

Program of Study Committee:
Eve Syrkin Wurtele, Co-major Professor
Leslie Miller, Co-major Professor
Julie A. Dickerson
Basil Nikolau
Shashi K. Gadia

Iowa State University
Ames, Iowa
2008

Copyright © Jie Li, 2008. All rights reserved.
TABLE OF CONTENTS

ABSTRACT iv

CHAPTER 1. GENERAL INTRODUCTION 1

1.1 Introduction 1

1.2 Literature review 2

1.3 Thesis organization 13

References 14

CHAPTER 2. METNETDB: A PLANT BIOLOGICAL NETWORK DATABASE BASED ON A LABELED GRAPH MODEL 20

Abstract 20

2.1 Introduction 21

2.2 Results and Discussion 26

2.3 Summary 50

Supplemental Data 50

Acknowledgments 56

References 56

CHAPTER 3. A GRAPH QUERY SYSTEM FOR BIOLOGICAL NETWORKS DATABASES BASED ON SUBGRAPH EXTRACTION 63

Abstract 63

3.1 Introduction 63

3.2 Related works 65

3.3 The MetNetDB storage model 71

3.4 Definition of subgraph extraction 74

3.5 Exact subgraph extraction algorithm 75

3.6 Extension to approximate subgraph extraction 84
ABSTRACT

To understand the cellular functions of genes requires investigating a variety of biological data, including experimental data, annotation from online databases and literatures, information about cellular interactions, and domain knowledge from biologists. These requirements demand a flexible and powerful biological data management system. MetNetDB is the biological database component of the MetNet platform, a software platform for Arabidopsis system biology. This work describes a labeled graph model that addresses the challenges associated with biological network databases, and discusses the implementation of this model in MetNetDB.

MetNetDB integrates most recent data from various sources, including biological networks, gene annotation, metabolite information, and protein localization data. The integration contains four steps: data model transformation and integration; semantic mapping; data conversion and integration; and conflict resolution. MetNetDB is established as a labeled graph model. The graph structure supports network data storage and application of graph analysis algorithm. The node and edge labels have the same extension capability as object data model. In addition, rules are used to guarantee the biological network data integrity; operations are defined for graph edit and comparison.

To facilitate the integration of network data, which is often inaccurate or incomplete, a subgraph extraction algorithm is designed for MetNetDB. This algorithm allows subgraph querying based on user-specified biomolecules. Both exact matching and approximate matching with biomolecules in networks are supported. The similarity among biomolecules is inferred from expression patterns, gene ontology, chemical ontology, and protein-gene relationships. Combined with the implementation of Messmer’s approximate subgraph isomorphism algorithm, MetNetDB supports exact and approximate graph matching.
Based on the MetNetDB labeled graph model and the graph matching algorithms, the MetNetDB curator tool is built with several innovative features, including active biological rule checking during network curation, tracking data change history, and a biologist-friendly visual graph query system.
CHAPTER 1.  GENERAL INTRODUCTION

1.1 Introduction

Many high-throughput technologies have been developed in recent years to detect and measure the cellular activities, which provide a basis for systematic study of plant functions. The enormous experimental datasets, such as transcriptomic, proteomic, metabolomic data and metabolic flux data, enable us to construct the genome-scale hypothetical interaction networks that occurred in the cells. Each type of these datasets contributes our knowledge of one type of cellular functions. For instance, the gene expression pattern can reveal the possible gene regulation mechanism.

The revolutionary of high-throughput biotechnologies would not be possible without the parallel development of bioinformatics and computational biology. Traditional comparative sequence analyses had played key roles in understanding functions of individual genes or proteins. Meanwhile, the comprehensive system-level study of genomics, transcriptomics, proteomics and metabolomics are emerged to understand how the genome encodes the cellular function of whole cell and organism. The enormous available experimental data, which provides the possibility of those 'omics, requires powerful systems biology software suits to help biologists to retrieve the underlying interactions. Therefore, MetNet (Wurtele et al., 2003; Wurtele et al., 2007) is developed as an integrated, open-sourced systems biology software platform for Arabidopsis data. The platform integrates a variety of powerful tools to handle different biological data, including experimental data, annotation data, biological network data and literatures. In the MetNet platform, Cytoscape/FCModeler (Dickerson et al., 2003) is the network visualization and modeling tool; BirdsEyeView the network navigation tool; PathBinder (Ding, 2003; Ding et al., 2005) performs text mining on Arabidopsis literatures; PubMed Assistant (Ding et al., 2006) allows biologists to query literature in PubMed, Google scholar and Google web search; MetaOmGraph can visualize and compute the expression data; exploRase (Lee et al., 2004) provides unique high-dimension data visualization features for experimental data;
AtGeneSearch and AtMetaboliteSearch are two web interfaces for gene and metabolite annotation aggregated from various data sources. To facilitate the development of MetNet tools, MetNetDB is designed to integrate biological network data and annotation from a variety of databases and biologists. These data is exported to other MetNet software tools through HiveMind (Johnson, 2005) service.

This type of biological data integration, though have been studied extensively for biological data (Davidson et al., 1995; Stein, 2003; Hernandez and Kambhampati, 2004), is complex and confronted with new challenges for biological network integration. Therefore, we introduce a labeled graph model to accommodate existing data into MetNetDB. Besides the flexibility of storing additional data from external data sources, the labeled graph model allows a variety of innovative features for the MetNetDB curator tool to be implemented easily, such as biological rule checking for data integrity, tracking data history for data provenance, and a visual pathway query system for network data comparison. In addition, I develop a subgraph extraction algorithm, which allows querying networks based on user-specified biomolecules. This algorithm, combined with my implementation of Messmer’s subgraph isomorphism (Messmer and Bunke, 1993; Messmer and Bunke, 1998), can solve generic biological network alignment problems.

1.2 Literature review

1.2.1 Biological network databases

A number of biological network databases have been created in the past. These network databases can be classified as pathway databases and molecular interaction databases. Pathway databases contain validated pathways either from literatures or from other sources verified by curators. Pathway databases can be further classified as metabolic pathway databases and regulatory pathway databases based on the focuses of the data set. BioCyc (Karp et al., 2005) and AraCyc (Mueller et al., 2003) are two examples of metabolic pathway databases. BioCyc has a collection of network databases of over
200 species. It provides comprehensive metabolic network data and reference information. Its derivation, AraCyc contains biological network data of Arabidopsis. The databases of BioCyc are classified as three tiers according to the curation quality. Initially pathways are constructed by using the PathoLogic algorithm (Karp et al., 2002). The algorithm evaluates the confidence of each pathway by scoring enzymes based on the occurrence patterns in the genome. Once verified by the curator, the evidence codes of interactions or other biological entities are adjusted if necessary.

AGRIS (Palaniswamy et al., 2006) and PathoPlant (Bulow et al., 2004) are two examples of regulatory pathway databases. AGRIS contains information about Arabidopsis promoter sequences and transcriptional factors (TF) from literature and gene annotations. Those documented relationships between one TF and its target gene are then added to the regulatory network database. PathoPlant stores signal transduction pathways related to plant pathogenesis. PathoPlant extracts the signal transduction parts from TRANSPATH (Krull et al., 2006) and extends it with the interactions between plants and pathogens.

It should be noted that, to get a comprehensive biological context, current biological network databases normally do not limit their data set in one type of biological networks. Some databases like KEGG (Hashimoto et al., 2005) have both plentiful metabolic pathways and some regulatory pathways. KEGG pathway database has a general reference network database and species-specific network databases. It contains protein-protein interactions, metabolic networks, and genetic regulatory networks. In addition, the pathway diagrams are manually drawn to cater to biological tastes. Combined with KEGG gene and ligand database, the data can also be used for modeling and simulation purposes. In addition to the pathway database, KEGG BRITE provides a knowledge hierarchy database containing relationships between biological entities.

Unlike pathway databases that have large-scale connected network data, molecular interaction databases contain pair-wise molecular interaction data (e.g. protein-protein interactions). The interactions give clues how biological molecules affect or regulate each other. Thus, the study of
these interactions can lead to new pathway construction and reveal gene or protein functions. BIND (Alfarano et al., 2005) and DIP (Salwinski et al., 2004) are two examples of interaction network databases. BIND contains interactions between molecules and complexes while DIP focuses on the experimentally verified protein interactions. Both databases have abundant annotation of interactions and links to external databases.

Compared to the validated pathway databases, those molecular interaction databases collect data from high-throughput experiments and thus have larger data set. However, the interactions usually have small size. They cannot be used in modeling and simulation purposes directly. CellCircuit (Mak et al., 2007) tries to bridge the gap between two types of data. It constructs hypothetic protein network models from publications. Each model in the database is scored by Gene Ontology (Harris et al., 2004) annotation. Similar publications of models are also scored according to the overlap between models of each publication and interactions of each model. The models allow computational biologists to evaluate and validate the hypothesis more easily.

Some biological network repositories are not constructed on database technology but are still valuable to the biologists. Biocarta (http://www.biocarta.com/) is an example of these repositories. Biocarta has a collection of pathway diagrams. The pathway diagrams are maintained by ‘pathway gurus”. Biologists can use vector graphics drawing tools to draw a new pathway and submit the picture to the repository. Biologists can also comment the diagrams online. The old version of pathways can be retrieved if necessary.

Although there are a lot of interaction network databases available for biological studies, most of them are focused on one purpose. For instance, AraCyc originally focused on the metabolic networks of Arabidopsis, and until recently, begin to integrate some regulatory networks; AGRIS focuses on transcriptional regulation of Arabidopsis. KEGG focuses on various types of networks for many organisms, but has limited annotation of biomolecules that participate in biological interaction networks. Besides network databases, many databases focus on the annotations, experimental data,
and literature of biomolecules. It would be helpful to integrate relevant information of biomolecules into networks to provide a single view for analysis tools and biologists.

1.2.2 Data integration

In past decades, database technology has been widely applied on the biological data management. These biological databases can be categorized according to their purpose, such as sequence databases, structure databases, pathway databases, experiment databases, annotation databases, literature databases, and many species-specific databases. These databases usually contain three layers (Stein, 2003): database layer, business logic layer and user interface layer. Most databases also provide bulky download of whole data set publicly. However, since these databases are focused on one subset of biological knowledge, it is necessary to provide an integral view for biologists to get the comprehensive knowledge of one particular biological topic.

Currently, there are three ways to integrate biological data: link integration, view integration and data warehouse (Davidson et al., 1995; Stein, 2003; Hernandez and Kambhampati, 2004). These integration approaches are different in the data storage and degree of integration. Link integration and view integration do not store the data locally, while data warehouse approach retrieves a specific subset of data into the local database. In addition, view integration tightly integrates external data while link integration only provides linkage information among data entries in different data sources.

Link integration is very common in databases with web interfaces. Usually there are cross-references among different databases for a biological entity. Biologists can follow the hyperlink to navigate the detail annotation among those databases. SRS (Etzold and Argos, 1993; Etzold et al., 1996; Etzold and Verde, 1997; Zdobnov et al., 2002, 2002) is a system specific to this type of integration. SRS means “sequence retrieval system”, which is originally designed for accessing various biological sequence databases. SRS has linked different data entries among these databases. Therefore, annotations from different databases are aggregated into a single result web page. Link
Integration has been adopted by many biological databases because of the simple implementation and increasing cooperation among database maintainers. However, this type of integration usually needs stable link information. If an external database changes the link for one data entry, the local system has to update the corresponding data entry manually. Since data is stored in external databases, the link integration cannot provide powerful query interfaces except the keyword available in external databases. Moreover, it is hard to extend the data, especially if biologists want to integrate their own knowledge.

View integration uses an interface to collect user queries. These queries are analyzed and converted to queries to external databases. The query results are aggregated for the user. This type of integration does not store data locally either. DiscoveryLink (Haas et al., 2001) is an example of view integration. This approach relies on a wrapper for individual database that can map the external data sources into the local data model. The query system parses the user request and transforms it into a set of queries for relevant wrappers and external databases. Then, the query results are combined into a single result for users.

With the technology development, web service of biological data repository is emerged as a new integration approach. Web service can be regarded as a special type of view integration, in which the data sources participated in the integration efforts. BioMoby (Wilkinson and Links, 2002; Wilkinson et al., 2005) is an example of biological web service. Compared to traditional web services, BioMoby predefines ontology for XML schema. Many tools are developed for creating and consuming BioMoby web services. In addition, a handful of databases registered for BioMoby web services. For example, Arabidopsis web services are available at http://bioinfo.mpiz-koeln.mpg.de/araws.

Web service is a deep cooperation among biological databases. Compared to the traditional view integration approach, the data providers participate in the data integration efforts by provides a simple and powerful data access interface. In most cases, the web service consumers do not need to learn the detail data organization of external data sources. Web service is promising for many data analysis
tools if these tools only retrieve and do not modify external data. However, web service, like link and view integration, is also affected by the external factors like network connection and database availability. Moreover, the performance is questionable for large-scale data analysis.

In the database-warehouse approach, a subset of or complete set of external database is copied to the local database. Because all data are located in the local database, the database is not vulnerable to external factors, such as network connection, database maintenance, etc. The database can optimize queries and process data locally according to a single data schema. Curators can add additional annotation to existing data. Because of these advantages, MetNetDB uses this strategy for data integration. However, there are challenges in this type of data integration. The integration should solve problems in data model transformation and integration, semantic mapping, data conversion, conflict resolution (Davidson et al., 1995; Hernandez and Kambhampati, 2004). The detail is presented in Chapter 2.

1.2.3 Graph data model

To solve the problem of data model transformation and integration, MetNetDB needs to find a data model that can accommodate available data. Existing data models are usually developed either to serve as a storage model or to provide the basis of some aspects of pathway analysis. In addition, these data models have a variety of specific limitations. For instances, relational database models are only useful when the data can be represented as a set of tuples. It requires another data model above the databases to utilize the data to simplify the use of the data. Computational models are only useful for mathematical analysis like metabolic flux analysis of a biological process. Graph-based models are natural representation of biological networks. However, existing graph-based models like chemical compound graphs or interaction graphs (Deville et al., 2003) are limited because of ambiguous representation caused by omission of the interactions or metabolites, respectively. Bipartite graph and hyper-graph models (Deville et al., 2003) cannot represent metabolic networks
and regulatory networks at the same time. Objective representation models (Deville et al., 2003) are useful for storing various types of data and easy to extend. However, it is not a graph model, and thus requires additional programming to interpret them as graphs when graph-based analysis needs to be applied on the data. Therefore, it has same problems as the relational model in this sense.

To combine the natural representation and analysis power of graph model and flexible extension capability of object data model, MetNet labeled graph model is designed to represent the biological network. In addition, rules and operations are added to the labeled graph model. Similar to the relational database constraints, rules are used to guarantee the biological data integrity of the labeled graph model. Operations are used to define the possible graph edit operations. The labeled graph model supports us to implement all features in the curator tool and fulfill the goal of MetNetDB. The detail of MetNetDB labeled graph model is presented in Chapter 2.

1.2.4 Pathway query

Unlike conventional sequence databases, network databases must handle the graph data type, which is complex and poses challenges on data integration. For instance, in a sequence database, once the data entry is determined according to some identities like locus ID or GenBank accession number, the annotation from external databases can be associated with the data entry in the local database easily. However, it is hard to map networks from external data sources with a network in local databases directly because there is no practical identity mechanism to identify a network. In addition, because of the limited biological knowledge about interaction networks in cells, the network data is usually inaccurate and incomplete. This becomes more complicated while biologists use different terms to describe same biomolecules. Therefore, we need a solution for comparing graph data, including interactions, pathways and networks.

Pathway query means finding pathways from a database given a query condition. The query condition can be the existence of biomolecules, interactions or a structure in a pathway. Almost all
biological network databases provide some capabilities of pathway querying. In addition, some software tools have been developed specific for pathway query. However, there is not a perfect solution for pathway query until now.

For those biological network repositories without a comprehensive data model, the network data has to be compared and curated manually. For instance, Biocarta use images to visualize the graph. This put many challenges on data sharing. Data submitters have to look through existing pathways to determine if the proposed pathway is already in the data repository. Then, the submitter draws the graph by using graphics tools. In addition, the hyperlinks in the image have to be created manually. The pathway data can only be browsed or queried by pathway names or gene names. Thus, such types of repositories cannot provide same powerful features as those biological network databases.

Many biological network databases use web interfaces to implement the network tools. Web interfaces are based on HTML and they are prevalent in conventional genome annotation databases. The web interfaces can display texts, images and hyperlinks. However, a graph data structure is not easy to be represented by this approach. For instance, Biosilico (Hou et al., 2004) only lists reaction lists of a pathway. It is hard for biologists to understand the topology structure of the pathway. UMBBD (Ellis et al., 2006) manually arranges texts of biological entity names and uses ASCII characters cleverly to represent the directed edges. However, biologists cannot query and interact with the biological network easily. This representation practically prohibits large-scale data submission. MONOMICS (Dunin-Horkawicz et al., 2006) uses Graphviz (Ellson et al., 2002) to generate images of pathway graphs and then put the images on the web. CellCircuit also uses images and texts to visualize hypothetic networks. All these tools lack of adequate capabilities for data manipulation.

To meet the cross-platform requirement and provide graph manipulation features, many network tools are implemented as Java or LISP applications. PathCase, PaVESy (Ludemann et al., 2004), PATIKA (Demir et al., 2002; Nisanci, 2003; Dogrusoz et al., 2006), NetMatch (Ferro et al., 2007), MetaPathwayHunter (Pinter et al., 2005), SAGA (Tian et al., 2007) and BioCyc pathway tool (Karp
et al., 2002) are typical examples of this type of network tools. All these pathway tools have graph visualization, edit or query features. However, most of them are designed for special purposes. Therefore, they have limited graph query capabilities. The detail of these tools is discussed in Chapter 3.

1.2.5 Subgraph isomorphism

In MetNetDB and other biological network database, the input query graph used to query the database is usually smaller than the networks in the database. This type of query is a problem of subgraph isomorphism in graph theory. Subgraph isomorphism is trying to match one graph to the subgraph of another graph, and therefore, is a very powerful tool in biological network comparative analysis. An example of subgraph isomorphism is shown in Figure 1. Suppose there are two graphs $G$ and $H$, if we are trying to find a one-to-one correspondence between the node set of $G$ and the node set of $H$, we called this mapping as graph isomorphism. If one graph, for example $J$, has more nodes than the other, we are trying to find the subgraph isomorphism from $G$ to $J$. This means, we are trying to find a subgraph $S$ of $J$, such that $G$ and $S$ have a graph isomorphism. If the distortion of nodes and edges is allowed in the subgraph isomorphism, this is called as error-tolerant subgraph isomorphism (Messmer and Bunke, 1998).

If all biological network data is accurate and complete, the graph matching between an input graph and existing graphs in the database can be solved by comparing the labels in the node and edge of graphs. Since the node label in most biological networks are distinct, the graph matching for biological networks usually can be performed in polynomial time. However, current available biological networks usually are incomplete or sometimes inaccurate. In addition, it would be valuable if approximate matching were applied to the existing biological networks in comparative study. Therefore, error-tolerant graph matching needs to be used to find out potential matching graphs in the graph databases.
Figure 1 Graph isomorphism and subgraph isomorphism. The graph $G$ is matched to the graph $H$ and the graph $J$. $f$ is the graph isomorphism from $G$ to $H$. $k$ is the subgraph isomorphism from $G$ to $J$ since there is a subgraph $S$ of $J$ such that $G$ has a graph isomorphism to $S$.

(Sub)Graph isomorphism algorithms have been studied more than 30 years. It is well known that any known graph isomorphism algorithms requires an exponential computational time in the worst case. Many algorithms use heuristic approaches to reduce the search space. Corneil (Corneil and Gotlieb, 1970) introduces a depth-first backtracking search algorithm by comparing one node to the nodes in another graph recursively. Ullman (Ullman, 1976) considers the adjacent nodes of the matching node to do the forward checking, which has a better performance than the original backtracking algorithm. Falkenhainer (Falkenhainer et al., 1989) tries to convert two matching graphs to an associated graph. Thus, the isomorphism problem is converted to maximal clique detection in associated graphs. Nilsson (Nilsson, 1980) uses A* algorithm to compute the cost of graph edit operations which convert one graph to the other graph. This is particular useful when we compare two graphs in an approximate way.

These graph isomorphism algorithms only consider the matching between two graphs. This is inefficient if we want to compare one input graph with graph models in a graph database if those graphs share common structures. Messmer (Messmer and Bunke, 1993; Messmer and Bunke, 1998) presents an inexact subgraph isomorphism which aimed to the query for a graph database with approximately matching in nodes and edges. If graphs in the graph database, called as model graphs, contain many common substructures, Messmer’s algorithm can save a lot time by comparing these
common substructures only once. This is achieved by an offline decomposition step. In this offline step, all model graphs in the database are decomposed sequentially and the substructures are organized in a decomposition tree. Therefore, a model graph, if contains a substructure in the decomposition tree, will be represented as the summation of this substructure and the remaining structure. The remaining structure is recursively decomposed by existing substructures in the decomposition tree until no one exists. At this point, the unmatched remaining part becomes a substructure in the decomposition tree for future decomposition of other graphs.

Messmer's algorithm may be improved by applying domain knowledge to decrease search space. Fuchs (Fuchs and Le-Men, 2000) introduces a way to incorporate a priori knowledge (i.e. matches between some elements of the input graph and model graphs are already known) into the algorithm to decrease the search space. Similarly, MetNetDB applied biological knowledge on the algorithm to decrease the number of nodes in comparing graphs. For example, a gene is never matched to a metabolite.

Subgraph isomorphism is an essential step for biological network query. However, to get the biologically meaningful alignments between networks, we have to get the common substructure between two graphs. Therefore, we created a subgraph extraction to get the relevant substructure from isomorphism. The detail of subgraph extraction is discussed in Chapter 3.

1.2.6 Network curator tool

Almost all biological databases have tools to help data curation. For network databases, a curator tool must support features like graph visualization and edit. Besides these essential features, the curator tool must address the problems for biological databases, especially data provenance (Jagadish and Olken, 2004). Data provenance means saving the change history of data entry. Unfortunately, because of the inherent complication of biological network data, none of existing databases and curator tools support this feature. In addition, because of the limited qualified curators available for biological
network databases, the curator tool should support network curation efficiently. The MetNetDB curator tool implements several innovative features to address the problems. The detail of the curator tool is discussed in Chapter 4.

1.3 Thesis organization

This dissertation consists of five chapters. Chapter 1 is the general introduction; Chapter 2, 3 and 4 are three manuscripts to be submitted to Plant Physiology, BMC Bioinformatics, and Bioinformatics; Chapter 5 is the general conclusion. In the general introduction, I present the general background of biological data integration problem. The three manuscripts are all from my MetNetDB research project. Chapter 2 is about data integration approach in MetNetDB development, in which I identify the problems in biological network data integration and provide a generic approach for the integration specific to network databases including MetNetDB. Heather Babka provided data for MetNetDB and comments about the database and the curator tool. Nick Ransom implemented AtGeneSearch that allows querying the gene annotation collected in MetNetDB. He also wrote an application to export pathway data to other MetNet tools. Yves Sucaet implemented the MetNetDB web service and scripts for exporting data to SBML and BioPAX. I implemented the database, including the data collection, data integration, and data service for MetNet platform. Chapter 3 is about a graph query system based on my subgraph extraction algorithm, in which I provide a solution for biological network comparison and query. I developed the subgraph extraction algorithm and implemented the visual graph query system. Chapter 4 is about the features of MetNetDB curator tool. In this paper, I presented some innovative features and ideas behind these features specific to the curation of biological network data. Heather Babka provided user requirements and comments from the perspective of a biologist, which is critical and fundamental for the curator tool. Nick Ransom implemented part of the curator tool and provided comments of usability from the perspective of a beginner. Tian Xia investigated and implemented part of the graph-matching algorithm, which helped
me evaluate and determine the best approach of the algorithm implementation. He also collected a
variety of graph data, tested implementation and reported the bugs. In the conclusion part, I
summarize the general solution for biological network data integration and provide some future
directions and applications of MetNetDB and the graph matching implementation.

References

Alfarano C, Andrade CE, Anthony K, Bahroos N, Bajec M, Bantoft K, Betel D, Bobechko B,
Boutilier K, Burgess E, Buzadzija K, Cavero R, D'Abreo C, Donaldson I, Dorairajoo D,
Dumontier MJ, Dumontier MR, Earles V, Farrall R, Feldman H, Garderman E, Gong
S, Magidin M, Moniakis J, Montojo J, Moore S, Muskat B, Ng I, Paraiso JP, Parker B,
Nucleic Acids Res 33: D418-424

interactions. In Silico Biol 4: 529-536


Comput Biol 2: 557-572

PATIKA: an integrated visual environment for collaborative construction and analysis of
cellular pathways. Bioinformatics 18: 996-1003


CHAPTER 2. METNETDB: A PLANT BIOLOGICAL NETWORK DATABASE BASED ON A LABELED GRAPH MODEL

A paper to be submitted to *Plant Physiology*

Jie Li, Leslie Miller, Heather L. Babka, Nick Ransom, Yves Sucaet, Julie A. Dickerson, Eve Syrkin Wurtele

Abstract

MetNetDB is a biological network database for Arabidopsis and other plant species that is designed to encompass and integrate metabolic and regulatory interactions. The combined biological network is stored as a labeled graph model to facilitate integration of known or hypothetical biological interrelationships from multiple data sources. Biological entities and interactions are represented as nodes, and the relationships between them are represented as edges. Biological properties of entities (synonyms, entity type, subcellular localization, literature citations, gene annotations, and metabolite formula) and interactions (interaction type, reversibility, and EC number for enzymes) are stored as corresponding node labels. Stoichiometric and kinetic parameters for interactions are captured as edge labels. A separate curator tool supports searching, visualizing and general manipulation of the networks. The history of each change is retained. These novel features are crucial for biological network integration and also provide the structure for graph-based search and analysis of the network structure. MetNetDB networks can be shared through Cytoscape-compatible XML files or a dedicated API. MetNetDB serves as the primary data repository for the MetNet suite of visualization and analysis tools.

Availability: http://www.metnetdb.org/MetNet_db.htm
2.1 Introduction

Biological network databases can provide a platform for integration, and viewing of high-throughput transcriptomic, proteomic and metabolomic data, literature evidence, and annotation together with the biological networks they are associated with. In addition, they can be designed to enable computational analysis and modeling of experimental data in the context of known and/or hypothesized biological network(s). Such analyses facilitate the development of experimentally-testable hypotheses concerning the functions of biological molecules, metabolic or regulatory interactions, or network structures.

Unlike sequence or molecular structure databases, databases for biological networks must store data with complex internal relationships. There are multiple types of biological entities (e.g., DNA, RNA, polypeptides, proteins, and metabolites), hybrids of these entities (e.g., acetylated histones, methylated DNA, proteoglycans, and glycoproteins). Many types of interactions can occur between these entities (e.g., catalysis, transport, complex formation, allosteric inhibition, transcriptional regulation) with a wide range of kinetic and stoichiometric properties. Adding to this inherent complexity, many aspects of the network are uncertain, and there is a tremendous amount of missing data. Even in Arabidopsis, a model species, the precise functions of most genes are not yet understood (Swarbreck et al., 2008); regulatory interactions are even less comprehensively understood, and kinetic information is rare.

A biological network database can capture and represent biological interrelationships in many ways. Its data storage model should be rich enough to describe complex relationships and uncertainties inherent in the network. To take advantage of continuously expanding experimental technologies (e.g., ChIP-ChIP (Wu et al., 2006), protein localization (Sadowski et al., 2008), laser desorption-GC-MS (Chen, 2008)), the model needs to be flexible enough to incorporate not only the current types of biological data, but also to accommodate new data types. The model also should be
able to store, manipulate and export biological network data easily, and to incorporate new information from existing data sources effortlessly.

Biological network databases have been implemented as an object data model, a frame data model, or, several types of graph models (Table I). The object data model is based on object-oriented programming principles, and stores data as a collection of objects (Booch, 1994). Biological concepts are described through class-definitions and individual units are represented as objects (e.g., molecules, interactions, pathways, networks). The object data model suffers because it is a collection of objects with minimal connectivity between the objects. The frame data model (Reimer and Hahn, 1983; Karp and Paley, 1996) is a collection of frames, which represent knowledge; frames can be either a class (e.g., a concept like “polypeptide”) or an instance (e.g., a concrete biological entity like “acetyl-CoA”). The frame model use slots to define the relationships among frames. However, the frame model, similar to the object data model, cannot be used in the graph-based analysis directly. Graph models are based on nodes and their interconnecting edges (Cormen, 2001). Graph models have an added advantage over object and frame data models: a graph implicitly represents a network (a network can only be represented by object or frame data models artificially, at best). Thus, the graph structure provides a natural way for representation and manipulation of chemical structures and biological networks. A variety of graph-based approaches have been developed to represent the complexities of data content (Renzo and Claudio, 2008). Several graph models have been developed for computational analysis of biological and chemical phenomena, including chemical compound graphs, chemical reaction graphs, bipartite graphs, and hyper-graphs (Deville et al., 2003). Each of these graph models is designed for a specific type of computational analysis. Simple graph models contain nodes and edges which can carry little information; the edges can be directed or undirected. The simple graph model is the basis of all other types of graph models and its extremely simplicity allows the focus of graph-based analysis. Chemical compound graphs, reaction graphs, bipartite graphs and hypergraphs provide a slightly richer structure to highlight specific information such as
compounds, reactions or topology relationships for a specific analysis, but do not work as a means of storing complete network data. Hypernode graph model used in PATIKA (Demir et al., 2002; Demir et al., 2004; Dogrusoz et al., 2006) allows the nodes to represent graphs. The model is specifically designed for multiple levels of abstraction for pathways and molecular complex state transitions (such as transport, and modification). It allows storage of pathway knowledge even some is incomplete. However, it lacks features for data integrity such as constraints from a graph-based perspective. To expand the types of information that can be represented, SAGA (Tian et al., 2007) uses an attribute graph model, in which the nodes contains attributes used in the graph matching algorithm. Although this representation is powerful enough to be searched, it currently cannot be used to store complete biological network knowledge such as stoichiometry and kinetic parameters.

<table>
<thead>
<tr>
<th>Model</th>
<th>Terminology</th>
<th>Subtype:Examples</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object oriented</td>
<td>Classes, objects</td>
<td>UM-BDD (Ellis et al., 2006)</td>
<td>Storage</td>
<td>Minimal connectivity between objects; limited visualization and analysis</td>
</tr>
<tr>
<td>(Booch, 1994); Database is collection of objects</td>
<td></td>
<td>aMAZE (Lemer et al., 2004)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td>Frames, slots, entries</td>
<td>BioCyc (Karp et al., 2005) Reactome (Vastrik et al., 2007)</td>
<td>Storage; semantic restrictions; some graph queries possible</td>
<td>Limited visualization and analysis</td>
</tr>
<tr>
<td>(Reimer and Hahn, 1983); Database is collection of frames</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph</td>
<td>Nodes, edges</td>
<td>Chemical compound graph and reaction graph (Deville et al., 2003)</td>
<td>Simplified models for analyze topology relationships among chemical compounds or reactions</td>
<td>Limited storage, hard to integrate various types of networks</td>
</tr>
<tr>
<td>Model</td>
<td>Terminology</td>
<td>Subtype:Examples</td>
<td>Advantages</td>
<td>Disadvantages</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Attribute:</td>
<td>SAGA (Tian et al., 2007)</td>
<td>Support storing attributes of biomolecules and interactions in the nodes for graph similarity computation</td>
<td>No edge labels, cannot store interaction properties; Limited storage</td>
<td></td>
</tr>
<tr>
<td>Bipartite graph,</td>
<td>Deville et al., 2003)</td>
<td>Can represent compound nodes and reaction nodes in same graph</td>
<td>Cannot represent metabolic networks and regulatory networks at same time without extension.</td>
<td></td>
</tr>
<tr>
<td>Hypernode graph model) PATIKA</td>
<td>(Demir et al., 2002; Demir et al., 2004; Dogrusoz et al., 2006)</td>
<td>Represent pathway knowledge in different abstract level. It is regarded as an extension of bipartite graph (Deville et al., 2003)</td>
<td>Lack of data integrity facility. Intractable to validate all possible pathways</td>
<td></td>
</tr>
<tr>
<td>Labeled</td>
<td>(MetNet, this paper)</td>
<td>Large-scale integration of networks computationally, Sophisticated graph-based analysis; Graph matching algorithms including inexact network matches; Semantic restrictions; Kinetic and stoichiometric data stored on edges; High network storage and retrieval capacity</td>
<td>Complex implementation. Lack of a formal definition of biological network oriented graph query language due to time constraints</td>
<td></td>
</tr>
</tbody>
</table>

While existing graph models are good for graph-based network analysis, they are not designed to serve as storage models. Here, we describe the MetNetDB labeled-graph model. MetNetDB labeled graph model is designed to store arbitrary information around the biological networks. It represents both molecular entities and interactions as nodes (multiple subtypes are used to distinguish molecule and interaction types). It also defines the graph operations and rules for data integrity. Meanwhile,
this storage solution can be used in the graph-based analysis flexibly. For example, MetNetDB labeled graph model supports the ad-hoc reconstruction of subnetworks of interconnecting entities through a $p$-neighborhood search.

The labeled graph model has been implemented for Arabidopsis, and is the central data repository of the MetNet project (Wurtele et al., 2003; Wurtele et al., 2007). MetNetDB stores biological networks and annotations for *Arabidopsis thaliana* and other species. In this paper, we introduce the labeled graph model and present an overview of the MetNetDB system. There are four main contributions to this work.

First, MetNetDB collects and integrates biological interaction networks and annotations from various data sources. These include metabolic networks, regulatory networks, protein subcellular localizations, and annotations of genes, proteins and chemical compounds. The resulting integrated data set is needed for biologists to understand the biological functions of genes, proteins and chemical compounds.

Second, we have created a labeled graph model for the integration of a biological network database. The model provides flexibility for database expansion and internal or third-party tool implementation.

Third, we have designed a MetNetDB curator tool that is implemented on top of the labeled graph model. This allows biologists to create new networks and to modify and update the existing data. In addition to curating annotation, the curator tool allows pathway structures to be compared and integrated. This last feature is especially useful for integration of pathways originating from different biologists and different data sources.

Finally, based on the labeled graph model, the MetNetDB database provides an enriched source of integrated Arabidopsis metabolic and regulatory network data. The data are accessible to all biological and computational researchers. MetNetDB also provides network data for the MetNet tools (Dickerson et al., 2003; Wurtele et al., 2003; Lee et al., 2004; Ding et al., 2005; Yang et al., 2005;
Wurtele et al., 2007; Mentzen et al., 2008; Mentzen and Wurtele, 2008) in a graph data structure directly through a HiveMind service (Johnson, 2005). The network data can also be exported as formatted data files, for example, XML or SBML files, depending on the needs of the biologists and/or tool developers. For Java and .NET application developers specifically, a dedicated web service is available for download through http://metnet3.vrac.iastate.edu/api.

2.2 Results and Discussion

A biological network database must integrate information from a variety of heterogeneous sources, and thus inevitably confront the inherent challenges of data integration (Davidson et al., 1995; Hernandez and Kambhampati, 2004). For example, different databases use different data models; these may use different terms to describe the same biological properties (e.g., the concept of sub-compartmental localization (say, thylakoid lumen) is identified as a component’s “location” in some databases, in other databases as its “compartment”). In addition, data often conflicts across databases, or even within a single database. The MetNetDB labeled graph model, designed as a “one size fits all” model, would allow the integration of data and data models from multiple sources feasible. We used a four-step approach to integrate data from multiple external databases. First, the data models used by the external pathway databases and other annotation databases that we are using are mapped to the labeled graph model or to labels. Second, the terminology and/or underlying semantics used in the different external databases to be integrated are mapped to a common terminology. Third, the data from the external databases are transformed according to the data model mapped defined in the first step. The transformations require the semantics of the incoming data is matched with existing data before being stored. Finally, data are checked for inconsistencies and conflict resolution criteria are enacted. While many issues are able to be resolved automatically, hands-on expert curation is required for others.

In the remainder of the paper, our application of this data model to MetNetDB is described.
2.2.1 The Labeled Graph Data Storage Model

Supporting systems biology research in metabolic and regulatory pathways requires a rich and flexible storing model. The model has to be rich enough to store the pathway and annotation data. In the case of the pathways, it must be rich enough to form a superset of the data models used in the online data sources (Supplemental Table S1). In addition, the model needs to be extensible to provide support for the evolving requirements of research.

Labeled Graph Model

The labeled graph model provides the advantage of both the graph model for analysis and the object data model or the frame model for data storage. Since it is a graph, many graph algorithms can be applied directly to this data model. Meanwhile, the node and edge labels can be used to store individual biological entities as flexibly as either the object data model or the frame model, since any biological property can be described as part of a label (attribute, value). In each label, the attribute element is a string to indicate the property name, while value element is also a string type which can hold any type of data even the data is not known yet. For example, if a new experimental protocol and corresponding data are available, they can be encoded as a XML format string and then be stored in the label. The underlying database only needs to support the string value storage and does not need to change the database schema to adapt to the new data and its data model. The data for the label can even be a dynamic behavior under a special biological event (e.g., the increased activity of a protein complex under the existence of a chemical compound if we do not know the detail mechanism of this behavior) if it can be described by SBML or MathML. Therefore, the labeled graph model provides a way to incorporate more static and dynamic information into the graph structure. This structure can be easily transformed to various computational models for further analysis.

We developed a labeled graph storage model based on directed graphs to support biological networks. For simplicity, we drop the word “storage” for the remainder of this presentation. Our
labeled graph model is defined by a 7-tuple $G = (V, E, L, f_V, f_E, O, R)$ in which $V$ is the node set, $E$ is the edge set and $L$ is the set of node and edge labels. Labels have the form (attribute, value). The two functions, $f_V$ and $f_E$, are the node label assignment function ($f_V : V \to L$) and edge label assignment function ($f_E : E \to L$), respectively. $O$ is the set of operations defined on the graph model. The edit operations can be node insertion, node deletion, label substitution of nodes and edges, edge insertion, and edge deletion. These five operations allow us to change the status of the graph. $R$ is the set of rules that define data integrity. The rules are obtained from either biological knowledge or graph requirements. Rules are defined as a set of Boolean functions mapping from nodes, edges, and labels to a Boolean domain, e.g., $r : V \times E \times L \to \{0,1\}$. The rules determine which combination of nodes, edges and labels are allowed or prohibited.

To illustrate the model, we use the final reaction in the pathway “ethylene biosynthesis from methionine” (Figure 1). Figure 2 shows an example of the labeled graph model instance for the pathway fragment shown in Figure 1.

**Figure 1** A step of the pathway “ethylene biosynthesis from methionine”. This step contains two interactions, EC 114.17.4 and the catalysis activity of the enzyme ACC oxidase.
Representing Biological Networks Using the Labeled Graph Model

In the labeled graph model, nodes and edges are the smallest units that can take associated label information. An entity with location, which represents a biomolecules with a subcellular location, is mapped to a node in the labeled graph and all of its properties are assigned to the label of that node. An interaction is also represented as a node in the model. The motivation has been to avoid having to deal with hyper-graphs, which are inherently more complex and will require transformation to simple graphs to make use of most existing graph algorithms designed to assist in the study of biological systems. In addition, this structure is similar to the structure used in markup languages such as SBML. Edges represent the relationship between an entity with location and an interaction. In the case of catalysis and enzymatic reaction, the catalysis interaction node can also be connected to the enzymatic reaction node. The biological properties of each interaction, like reversibility or strength are stored in the corresponding node label. The coefficients are stored in the label of the edge between
the entity with location and the interaction. The direction of each edge indicates the direction of that interaction.

**MetNetDB and Labeled Graph Model**

The labeled graph model does not place any restriction on the type of data that needs to be stored except that it expects biological entities and interactions. The properties of entities and interactions required are dependant on the specific implementation of the model. The MetNetDB implementation of the model is summarized in Table II.

<table>
<thead>
<tr>
<th>Components of biological interaction network</th>
<th>Representation in the labeled graph model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity with location</td>
<td>Node</td>
</tr>
<tr>
<td>Biological properties of entity with location (name, organism, subcellular location, data origin, entity type, synonyms, abbreviation, references, user comments, external database cross-reference, confidence; for loci: probe set IDs, Uniprot ID, AraCyc pathway names, GO terms, TAIR annotation, TAIR reference IDs, MapMan BIN ID, MapMan category, MapMan annotation, TargetP location, gene name and symbol, AGRIS regulation, and AGRIS reference; for metabolites: PubChem CID, CAS, PubChem synonyms, ChEBI ID, ChEBI name, formula, IUPAC ID, SMILES). (Currently, MetNetDB defines 7 entity types and 74 subcellular locations.)</td>
<td>Node label</td>
</tr>
<tr>
<td>Interaction (Currently, MetNetDB defines 39 interaction types, such as positive and negative regulations.)</td>
<td>Node</td>
</tr>
<tr>
<td>Biological properties of interaction (organism, data origin, interaction type, EC number, confidence, reversibility, references and user comments).</td>
<td>Node label</td>
</tr>
<tr>
<td>Relationship between entity with location and interaction</td>
<td>Edge between nodes</td>
</tr>
<tr>
<td>Stoichiometric coefficient, kinetic data</td>
<td>Edge label</td>
</tr>
<tr>
<td>Relationship between interactions</td>
<td>Edge between interaction nodes</td>
</tr>
</tbody>
</table>

### 2.2.2 Data Content in MetNetDB

MetNetDB provides a wide variety of data to support the study of systems biology in plants: such as metabolic pathways, transcriptional regulatory networks, gene annotations, protein localization information, and metabolite annotations. The individual categories are briefly examined. Table III defines some terminology used in MetNetDB.
Interaction Networks and Gene Annotations

Besides the networks input by curators, MetNetDB regularly integrates AraCyc (Zhang et al., 2005) information. AraCyc is a metabolic pathway database for Arabidopsis. The data integrated in MetNetDB includes metabolic pathways and unique genes assigned to the pathways. Annotations and references for genes, metabolites, enzymes and reactions are also included.

In addition, AGRIS (Palaniswamy et al., 2006) data is also included. AGRIS is a regulatory interaction database. It contains information on: binding site of promoters, loci of associated transcriptional factors, and corresponding references. These data are integrated into the MetNetDB network using the interaction types: direct transcriptional activation and direct transcriptional inhibition (Figure 3).

### Table III. Terminology used in MetNetDB

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>Biological molecule (e.g., gene, RNA, polypeptide, protein complex, metabolite, cis-element) or environmental condition</td>
</tr>
<tr>
<td>Entitywithlocation</td>
<td>An entity in a particular subcellular location, such as an organelle or a membrane. This concept is especially important for computational analysis of experimental data that contains information about subcellular concentration of entities</td>
</tr>
<tr>
<td>Location</td>
<td>Subcellular location, subcellular compartment (Locations currently in use are given in Supplemental Data Table S2)</td>
</tr>
<tr>
<td>Interaction</td>
<td>Biochemical interaction, or the relationship among entitywithlocations (Interaction types are shown in Supplemental Data Table S3)</td>
</tr>
<tr>
<td>Left, from</td>
<td>The left part of an interaction when it is written down. (The reactants, for a chemical reaction.)</td>
</tr>
<tr>
<td>Right, to</td>
<td>The right part of an interaction when it is written down. (The products, for a chemical reaction.)</td>
</tr>
<tr>
<td>Reversible</td>
<td>An interaction that is kinetically bidirectional</td>
</tr>
</tbody>
</table>
Figure 3 Transcriptional regulatory network with metabolic network. This figure displays a part of the catalase ascorbate glutathione pathway combining data from multiple sources. In this Cytoscape (Shannon et al., 2003) graph using MetNet plugins (Dickerson et al., 2003; Wurtele et al., 2007), yellow nodes represent the biological entities located in the nucleus; light yellow nodes, cytosolic entities; grey nodes, microbody; blue lines, catalyses and enzymatic reactions; green lines, positive regulation; dashed lines, low confidence data. This example shows the integration of AGRIS regulatory networks and existing metabolic networks in MetNetDB. AT4G23810 encodes a transcription factor (TF). When adding this AGRIS derived information, we created a positive transcriptional regulation interaction between the gene product of AT4G23810 and its target loci (AT1G20630, AT4G35090, and AT1G20620).

Annotation of Genes

MetNetDB contains gene annotation associated with the interactions and entities, such as function annotation and external database ID mapping. To expand the annotation, we integrate a full copy of Gene Ontology (Harris et al., 2004) and TAIR gene annotations (Swarbreck et al., 2008). The mapping between the locus name and probe sets of Affymetrix ATH1 and AG genome arrays is included. The UniProt IDs (The UniProt Consortium, 2007) of gene products are associated with the corresponding Arabidopsis loci. MetNetDB also integrates MapMan (Thimm et al., 2004) bin
annotations, which include gene annotations and functional categories of the gene products (See Supplemental Data Figure S1).

Subcellular Localization Data

MetNetDB includes subcellular localization data (Figure 4) from several sources. If other information is unknown, locations are assigned from the protein sequences based on sequence similarity (TargetP (Emanuelsson et al., 2000)). This method is superseded in MetNetDB by experimental information, when available, from Arabidopsis protein localization databases that collect data from a combination of experimentation, literature, and predictions. Each of these localization databases includes evidence and references (if available) for the localization. Thus, users can determine if the location of a gene product is based on a sequence-based prediction, a single experiment, or verified through multiple experiments in the literature. These databases are PPDB (Friso et al., 2004) and plprot (Kleffmann et al., 2006), which contain plastid proteins of Arabidopsis and other plants; AMPDB (Heazlewood and Millar, 2005), which contains Arabidopsis mitochondrial protein data; AraPerox (Reumann et al., 2004), which contains peroxisomal Arabidopsis proteins; and, AtNoPDB (Brown et al., 2005), which contains Arabidopsis nucleolar proteins. Currently, if a protein has location information obtained from these databases, the corresponding data source is stored in the comment field while the evidence is encoded in the location confidence field. For example, most protein location data in PPDB is obtained from experiments; thus, the location confidence is encoded as “Direct Confidence” in MetNetDB. To accommodate the location information from these databases, MetNetDB defines a hierarchy of more than 70 subcellular locations. These locations are predominately based on GO categories and existing protein localization databases but in a few cases, they are specialized for plants. This subcellular location list (Supplemental Data Table S2) can be expanded as required.
Figure 4 Localization information for a Pyruvate Dehydrogenase E1 alpha unit. This AtGeneSearch web page shows the annotation of the locus AT1G01090. There are two data sources of protein subcellular localization information. One is from experimental data, in this case the Arabidopsis mitochondrial protein database PPDB (shown in comment field). The other is the TargetP prediction.

**Annotation of Metabolites**

Metabolite information is collected from various chemical compound databases. Based on this cross-referenced identification of metabolites, users can obtain relevant literature and chemical properties for a given metabolite. The most comprehensive chemical compound databases are PubChem (Wheeler et al., 2007) and Chemical Abstracts Service (CAS) (Buntrock, 2001). MetNetDB maps metabolite names to data entries in PubChem (Figure 5) because PubChem contains comprehensive information including chemical structures. Since the CAS database contains literature on chemical compounds, we map the metabolites in MetNetDB to the relevant CAS registry numbers from the NCI chemical compound databases (Sitzmann et al., 2008). In the PubChem database, a CAS registry number may be listed as a synonym, but it is not explicitly indicated as a CAS registry number. The CAS registry verification rules (David, 1997) are used by MetNetDB to search PubChem and verify those synonyms that are similar to CAS registry numbers, and assign CAS numbers to metabolites.
<table>
<thead>
<tr>
<th>Query</th>
<th>CAS Registry</th>
<th>PubChem CID</th>
<th>ChEBI ID</th>
<th>AraCyc Unique ID</th>
<th>MetNet Entity ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NH3</td>
<td>7664-41-7 possible: 17655-31-1, 15194-15-7, 208990-07-2, 8007-57-6, 214478-05-4, 17778-88-0</td>
<td>222, 2826723</td>
<td>16134</td>
<td>AMMONIA</td>
<td>152201, 218</td>
</tr>
<tr>
<td>ADP</td>
<td>possible: 1172-42-5, 84412-16-8, 20398-34-9, 58-64-0, 2092-65-1, 18389-49-6</td>
<td>449789, 11963549, 448895, 6852187, 448310, 444564, 445864, 197, 449469, 6022</td>
<td>16761</td>
<td>ADP</td>
<td>191</td>
</tr>
<tr>
<td>GTP</td>
<td>possible: 56001-37-7, 86-01-1, 13573-18-7, 7758-29-4, 24315-83-1, 14127-68-5, 13845-36-8, 10380-08-2</td>
<td>9543458, 6839, 7058167, 448009, 762, 449095, 4369387, 445710, 963</td>
<td>15996</td>
<td>GTP</td>
<td>669</td>
</tr>
<tr>
<td>Oxygen</td>
<td>7782-44-7 possible: 12185-07-8, 14797-70-7, 58238-79-2, 80217-98-7, 1338-93-8, 80937-33-3</td>
<td>977</td>
<td>15379</td>
<td>OXYGEN-MOLECULE</td>
<td>890, 17213, 105688</td>
</tr>
<tr>
<td>Eicosane</td>
<td>112-95-8 possible: 10006-10-7</td>
<td>8222</td>
<td>32929</td>
<td>158586</td>
<td></td>
</tr>
<tr>
<td>Docosanoic acid</td>
<td>112-85-6 possible: 18990-72-2, 5331-77-1, 34303-23-6, 2636-16-0, 20259-31-8, 7211-53-2, 16529-65-0, 2489-05-6, 4499-91-6</td>
<td>8215</td>
<td>28941</td>
<td>156499</td>
<td></td>
</tr>
<tr>
<td>Octadecane</td>
<td>593-45-3</td>
<td>11635</td>
<td>32926</td>
<td>158584</td>
<td></td>
</tr>
<tr>
<td>Nonacosane</td>
<td>630-03-5</td>
<td>12409</td>
<td>7613</td>
<td>CPD-7940</td>
<td>24518</td>
</tr>
</tbody>
</table>

**Figure 5 Mapping IDs of metabolites.** The IDs used in MetNetDB, AraCyc, ChEBI, PubChem and CAS are mapped according to exact string matching. Since multiple metabolites may have the same synonyms, there may be multiple IDs for each metabolite. In addition, one metabolite may exist in different subcellular locations.

We also incorporate metabolite information from ChEBI (de Matos et al., 2006), an ontological classification for chemical compounds. MetNetDB provides possible mappings between the metabolites in MetNetDB and those in ChEBI by using an exact name match approach. Supplemental Data Figure S2 illustrates the annotation of the simple metabolite ethylene in MetNetDB.

**Interaction Types**

To capture a wide variety of possible biological interactions, MetNetDB currently defines 39 interaction types, including enzymatic reaction, catalysis, transportation (7 types), transcription, translation, positive and negative regulation (9 types for each), and composition relation. The interaction types are derived from GO with several modifications. A full list of interaction types (See Supplemental Table S3) can be viewed in the MetNetDB curator tool. To integrate metabolic networks with gene regulatory networks, MetNetDB uses transcription/translation interactions, RNAs and polypeptides. In addition, to represent the composition of molecular complexes, composition-
AND and composition-OR are used to connect polypeptides, protein complexes, and enzyme activities.

**AND-OR relations**

MetNetDB uses “and” and “or” relations to describe the compositional relationships among enzyme activities, protein complexes and polypeptides. The use of composition relations helps to integrate metabolic networks with regulatory networks.

The relationship between enzymes and genes is crucial for modeling and predicting the existence of biological pathways using analysis tools. For example, enzymes with the same name (i.e., the same biological activity) may catalyze reactions in different subcellular locations. Examples are ATP synthase (in the thylakoid and cristae) and triosephosphate isomerase (in the stroma and cytosol). These enzymes typically have different compositions, i.e., the constituent polypeptides are products of a variety of combinations of genes from one or more gene families. To efficiently represent the relationships among genes and an enzyme, “AND” and “OR” relations are used as the operators of the composition relation (Figure 6). “AND” means the several polypeptides or protein complexes form a new protein complex while “OR” means several polypeptides or protein complexes perform similar functions independently. Other databases use different methods to describe these relations. For example, Reactome (Vastrik et al., 2007) use “defined set” to describe the complexes and protein paralogs, which is a variant of hypernode concept (Renzo and Claudio, 2008) used in several graph databases. Compared to this multilevel graph data model, MetNetDB “flattens” hypernodes into pathway graphs by introducing the AND-OR relations as a parallel concept of biochemical interactions. This will simplify the application of many graph algorithms in the pathway analysis without losing biological information.
Figure 6 Composition relationships. A composition-OR relationship and a composition-AND relationship of pathway “Acetyl-CoA biotin network” are visualized in the Cytoscape with MetNet plugins. The ACL protein complex has two subunits: ACL-A and ACL-B (Fatland et al., 2005). Therefore, a composition-AND relation is used to represent it. ACL-A could be composed of any of three polypeptides: ACL-A1, ACL-A2, ACL-A3. Thus, a composition-OR is used to connect these polypeptides and ACL-A. Similarly, a composition-OR is used to connect ACL-B1 and ACL-B2 polypeptides and ACL-B. In this example, tan edges indicate composition-OR interactions, and brown edges indicate composition-AND interactions. Light yellow nodes indicate biological entities in the cytosol. Yellow nodes indicate biological entities are in the nucleus. Node shapes indicate the type of entity: diamond nodes represent polypeptides or protein complexes; rectangular nodes indicate RNAs; hexagon nodes indicate genes.

**Synonyms and Abbreviations**

Biomolecules often have multiple names/synonyms. Conversely, a given name may describe more than one biomolecule. Such inconsistencies cause problems in integrating network data and using of network data. For example, ACC can be used as the abbreviation for acetyl-CoA carboxylase, an enzyme required for de novo fatty acid synthesis, or for 1-aminocyclopropane-1-carboxylic acid, an intermediate in the synthesis of ethylene. In particular, the use of ontologies and controlled vocabularies are currently being considered. For instance, ChEBI is developing an ontology for chemical compounds. Genes, RNAs and polypeptides have associated locus IDs that form a controlled vocabulary set. Despite these efforts, many ambiguities remain.

To address the ambiguity issues related to naming, MetNetDB uses the unique IDs in standardized databases to reference each biomolecule during data integration if such IDs are available.
For instance, Arabidopsis locus IDs are used to identify Arabidopsis genes, RNAs, and polypeptides. UNIQUE-IDs from the BioCyc open chemical compound database are used, when available, to identify metabolites.

In addition, MetNetDB also expands the lists of synonyms of biomolecules to facilitate data integration. Most of the synonyms currently in MetNetDB are computationally extracted from online databases, such as BioCyc’s open chemical database, KEGG (Kanehisa et al., 2004; Hashimoto et al., 2005), ChEBI, PubChem, TAIR, and BRENDA. Other synonyms are added based on common abbreviations (for example, CoA = coenzyme A, so acetoacetyl CoA = acetoacetyl coenzyme A). Still other synonyms are added from the literature or manually extracted from on-line databases by curators.

**Literature References**

MetNetDB collects literature references from several sources. AraCyc and AGRIS provide references for interactions. ChEBI provides references for chemical compounds. TAIR provides references for genes and corresponding RNAs and proteins. Others references are input by MetNetDB curators. Moreover, the curator tool integrates with the PathBinder (Ding, 2003; Wurtele et al., 2007) to allow searching PubMed and adding selected biological references.

**User Comments**

Users can input comments on pathways, interactions and biological entities through the curator tool. The comments can provide additional annotation of genes, source of data, reliability, or any information that does not fit in existing MetNet fields.
2.2.3 MetNetDB Implementation

Data Integration

MetNetDB collects biological interaction network data and annotation data from a wide variety of data sources (Figure 7). All information collected in MetNetDB from external data sources is represented in the MetNetDB labeled graph model described in the previous section. For instance, in the case of data from AraCyc, the special slots such as “reaction-list” slot in “pathway” frame, “enzyme” and “reaction” slot in “enzrxn” frame, “product” slot in “gene” frame, “catalyzes” and “component” slot in “protein” frame, “left”, “right” and “enzymatic-reaction” slots in “reaction” frame are extracted first. These slots represent the relationships among pathways, interactions and biological molecules in AraCyc (as well as other BioCyc databases). They are used to construct an initial graph instance of the pathway for MetNetDB. The remaining slots are converted to labels for the corresponding nodes or edges in the MetNetDB graph model. Then, MetNetDB adds additional...
nodes and edges to the pathway instance. RNA, transcription and translation nodes are added between gene and polypeptide nodes of the pathway instance. A protein and its components are connected by a composition-AND relation. If the compartment of a protein activity is known, a transportation node is created. Similarly, the AGRIS data is first converted to a labeled graph pathway instance according to the interaction types: “activate”, “repress” and “unknown”. The remaining properties of each gene are placed in the labels of the nodes representing genes.

The annotation of genes, proteins, and chemical compounds is stored in the label of the nodes representing them. The integration of these annotations contains two steps: 1) identify the corresponding entities, interactions and pathways in MetNetDB. If there is no mapping, a new entity, interaction or pathway is created in the database; 2) identify the corresponding MetNetDB label to add the biological properties or annotations from the external databases. For Arabidopsis, locus IDs are used as identifiers of genes and polypeptides to find the corresponding records in MetNetDB. The locus IDs are also used to connect AGRIS networks and networks stored in MetNetDB. For chemical compounds, the UNIQUE-IDs from BioCyc’s open chemical database are used. If a UNIQUE-ID is not available for a particular chemical compound, the MetNetDB curator would review the names, annotations, and interactions to determine if two or more names describe identical metabolites.

As new interactions are added to MetNetDB, we can determine whether it is already in the database. This matching process is implemented by comparing the entities participating in the interactions. Unlike enzymatic reactions, the protein complex and its corresponding composition relations are determined only by the components of the complex. Ideally, we could determine whether a given pathway is in the database by comparing the interaction lists. Otherwise, the curator would need to use the pathway query tool described later to determine whether it matches (possibly partially) a pathway in MetNetDB.

To represent the relationships among genes, polypeptides, protein complexes and their functional activities, an “AND-OR” relationship is used in MetNetDB. These relationships mainly come from
data sources integrated into MetNetDB. In the absence of experimental data on the constituents of
protein complexes, composition relations are deduced in MetNetDB from computational inferences
and user inputs. For computational inferences, we assume that genes with similar sequences have
similar functions. Thus, an “OR” relation is used for polypeptides in the same subcellular location
that have a similar sequence. On the other hand, if the polypeptides have low sequence similarity, but
are annotated as participating in the same function, they are assigned an “AND” relation in the protein
complex. Similarity between genes is computed using TAIR’s BLASTX. Subcellular location
annotation is used to establish which polypeptides could interact. In addition to the inference methods
based on the sequence analysis, a curator can edit the AND-OR relation in the curator tool. The
curator can create composition relations like other interactions in MetNetDB. The interaction
direction is from the smaller biological entities to the larger biological entities.

Conflicts among data from various data sources need to be resolved. For example, both
Affymetrix and TAIR have data files that contain the mapping relationship between the probe ID sets
and the locus numbers. However, they are inconsistent because the mappings are inferred based on
different approaches. In this case, we have adopted the TAIR mapping file in MetNetDB. In other
cases, the curator will look at the detail of the data. Since the curation quality depends on the
knowledge of curators and conflict resolution criteria, the history of data changes will be saved in
MetNetDB for future review and evaluation.

Synonyms of biomolecules are useful for data integration. MetNetDB has several ways to add
synonyms. Besides those synonyms from online databases, curators can input synonyms using the
curator tool. The curator tool can also automatically generate synonym candidates from existing
synonyms according to a set of curator-derived rules based on standard biological abbreviations, such
as “replace PP with pyrophosphate”. The synonym “geranyl-pyrophosphate” can be obtained from
“geranyl-PP” using this rule. These candidates then can be accepted or rejected by the curator.
Pathway Visualization

In the MetNetDB curator tool, there are three steps in the pathway visualization pipeline: graph data processing, visual metaphor assignment, and graph layout.

Graph data processing changes the structure of the pathway for the purpose of visualization. For instance, in the MetNetDB labeled graph model, there is always an interaction node between two entitywithlocation nodes to represent the relationships among the interactions and participated entitywithlocations. However, it is confusing to show an interaction node that only converts one biological entity to another one. Thus, although this interaction node remains in the database, it is not displayed in the curator tool. Transcription, translation, and transport are examples of interaction types that are “trimmed” in the curator tool. The corresponding entitywithlocation nodes are shown connected by a single edge.

Colors, shapes, and other legends are used as visual metaphors to indicate biological properties. For instance, a light yellow background indicates an entitywithlocation is in cytosol. In this metaphor assignment step, the curator tool iterates through all of the node and edge labels (biological properties) in the graph model. Once a node label or an edge label matches one or more metaphor assignment rules, the corresponding graph properties are applied to the node or edge.

Graph layout is the third component of network visualization. Graph layout is currently the subject of much research and is not yet optimized for biological networks (Li and Kurata, 2005; Yang et al., 2005). For the curator tool layout, we use the JzGraph package (http://jzgraph.sourceforge.net/). This package implements a variant of the Dot layout algorithm (Gansner et al., 1993) developed at AT&T. The layout algorithm minimizes edge crossings resulting in a clean visualization of the biological networks being rendered.
Pathway Comparisons

The curator tool uses pathway comparisons to identify differences among two or more pathways. This is important for incorporating new pathways into MetNetDB, and for reviewing pathway changes. The curator tool supports two types of pathway comparisons in an exact match mode: matching all biological properties of biomolecules, and matching all properties but the subcellular location. The first approach helps the curator identify incremental modifications of a pathway. The second one helps the curator identify the different parts of MetNetDB pathways that have locations and pathways from other databases, such as AraCyc updates (Figure 8), which often do not contain subcellular location information for biological molecules participating in interactions and pathways.

![Pathway Diagram](image)

**Figure 8 Matching AraCyc data in new pathway integration.** Since AraCyc data may not contain location information, sometimes multiple interactions in different subcellular locations in MetNetDB can match an interaction from AraCyc update. For instance, the reaction from pyruvate to acetyl-CoA occurs in both the mitochondrion and the plastid. The MetNet curator tool compares the context of the interactions. In this example, the curator can determine the interaction in the mitochondrion should be matched with the AraCyc update based on the subsequent interaction of acetyl-CoA.

Since AraCyc data does not contain location information in many of its pathway data, multiple interactions in MetNetDB can match the interaction from AraCyc. For instance, in Figure 8, the interaction from pyruvate to acetyl-CoA can occur in both mitochondrion and plastid. In such cases, a context is introduced to resolve the ambiguity. The context is defined as the sub-network surrounding the ambiguous interactions. In this example, the adjacent interaction provides a context that can be
used to resolve the ambiguity. Since the successor interaction from AraCyc has a product citrate, we know the process from AraCyc should be aligned to the process from mitochondrion in MetNetDB. Conflicts that are more complicated need to be resolved by curators manually.

**Pathway Query**

A pathway query means querying a single pathway or several pathways from the database. Similar to other biological network databases, MetNetDB supports keyword based pathway querying. In addition, MetNetDB supports querying pathways based on user-specified biomolecules or the partial graph structure of a pathway (Figure 9). Querying MetNetDB based on user-specified biomolecules allows biologists to map experimental data onto the network in MetNetDB. Querying based on a partial structure allows MetNetDB to identify if (a part of) a new pathway to be integrated is already in MetNetDB. If a portion of the new pathway exists in MetNetDB, the curator can add new interactions to the existing pathway.

Both exact match and inexact match are supported in both query types. The graph matching has been implemented based on Messmer’s subgraph isomorphism algorithm (Messmer and Bunke, 1998). In addition, we have developed a subgraph extraction algorithm to support extracting a subgraph from a list of nodes such as genes, metabolites and extracting common substructure from the isomorphism results.

Figure 9 and Figure 10 show an example of pathway querying based on a partial structure. In Figure 9, the curator draws a simple structure involving three metabolites (L-methionine, S-Adomethionene, and 5-methylthioribose). Figure 10 displays the best matching alignment found in MetNetDB.
Figure 9 Visual pathway query. An example of an input window when a curator has drawn a partial structure for a pathway query.
Figure 10 Pathway query result. This window displays a matching result from the structural query shown in Figure 9. The nodes of the input graph are designated by a red box. The dotted lines indicate the alignment between these nodes of the input graph and the pathway “ethylene biosynthesis and methionine cycle”. For instance, a dotted line indicates that S-Adomet node of the input graph is aligned to the S-Adomet in cytosol (light yellow background) of the pathway. In this pathway we can find a highlighted cycle that inexact matches the input graph.

Active Rule-checking During Curation

Rules are a set of constraints in MetNetDB that guarantee data integrity in a biological context. For instance, “an enzyme is not a metabolite” is an example biological rule used in MetNetDB. In MetNetDB, rules are classified into two types: biological and graph rules. Biological rules are rules to make the biological network comply with conventional biological knowledge. Graph rules guarantee the basic data integrity in the context of the graph. Rules are automatically checked each time a curator submits data or edits a pathway. The rules are classified as mandatory or optional. Any
violation of mandatory rules will block the data submission. Errors will be highlighted with a red box in the curator tool. Conflicts with optional rules allow exceptional biological data to be saved and curated but provide a warning to the curator by placing a pink box around the expected errors. An example of error is “an orphan node or a dangling edge in the network”. Rules can be easily added or removed. Supplemental Data Table S4 displays all rules currently implemented in the MetNetDB curator tool.

**Tracking Data Changes and Concurrency Control**

Another important feature of MetNetDB is that it supports tracking of changes in the database. Unlike Meta-All (Weise et al., 2006), which only supports discrete version requested by the user, MetNetDB automatically stores every changes in a pathway when the pathway is saved. For instance, if an entity with location is deleted from the database, MetNetDB saves this deletion and any resultant modifications to interactions and pathways. In another example, if an interaction or a pathway is modified, the previous version can be retrieved and compared for reviewing the modification. In addition, removed data can be undeleted if necessary (Figure 11).
Figure 11 Data history browser. The data history browser window displays previous edit operations on curator-selected biological network data. If the operation is “delete”, then the “undelete & edit” button will be enabled. Otherwise, the “review & edit” button will be enabled.

In addition to its function of data provenance, tracking is also used for concurrency control. MetNetDB and its curator tool are designed as a distributed application, which allows multiple curators to operate on the database at the same time. As a safeguard, the curator tool prevents two curators from updating the same entity with location or interaction at the same time. The workflow shown in Figure 12 illustrates how concurrent editing is eliminated. When a curator retrieves data from the database, the corresponding version information is saved at the client side. Before the curator can update the data, the most recent version information is retrieved from the database again to make sure nothing has been modified by someone else since the curator initially retrieved the data. In this way, MetNetDB guarantees that curators will always update the data that they are reviewing.
Figure 12 MetNetDB concurrency control in the workflow of curation. MetNetDB uses version comparison to prevent multiple curators from editing different versions of same data. At any time, only one curator can update the database. Before an update can be made, the most current version information must be same as the one when the curator initially retrieved the data, thus ensuring that the curator is updating the correct version of the data.

Data Export

Currently, MetNetDB can export biological network data to MetNet tools in several ways: propriety XML formats, direct table dump, and programming interfaces via HiveMind service and web service. For instance, Cytoscape with MetNet plugins (evolved from FCModeler (Dickerson et al., 2003; Wurtele et al., 2007)) and MetNetVR (Yang et al., 2005) currently use XML files generated by MapBuilder for network visualization and analysis. To simplify and unify the data access methods for the MetNet integrated platform, we have implemented HiveMind services based on the MetNetDB labeled graph model. All MetNet tools in the platform can use existing graph libraries to manipulate the biological network data obtained from these services. The MetNet web service API (http://metnet3.vrac.iastate.edu/api) is available for two popular software platforms: Java and .NET. In both environments, a set of classes that refer to logical MetNetDB concepts are available. A tool for exporting to SBML format (Hucka et al., 2003) and BioPAX format (Luciano and Stevens, 2007) can be accessed at http://metnetdb.org/metnet3.
2.3 Summary

As a biological database, MetNetDB supports data import, edit and export. Currently MetNetDB contains three components: a database, a curator tool and a service interface. All of these are built on a labeled graph model designed for MetNetDB. By using the labeled graph model, the database can integrate and store complicated network data and annotation from heterogeneous data sources. In addition, the curator tool relies on the model to implement features like tracking data changes, validating data input, comparing and querying biological networks. The graph model also benefits other MetNet tools since they do not need handle relational model of the underlying database system. In this context, MetNetDB provides a comprehensive repository supporting systems biology research in Arabidopsis.

MetNetDB presents an example for biological network data integration. Biological network databases for other species can use similar approach to integrate a variety of data sources and to provide a single data view for pathway analysis tools. This approach includes data model transformation and integration, semantic mapping, data transformation, and conflict resolution. The inexact graph-matching implementation is based on the labeled graph model and can be used for network integration when these biological networks are represented as the labeled graph model.

Supplemental Data

The following materials are available in the online version of this article.

Table S1. Data sources of MetNetDB

Table S2. The subcellular location hierarchy defined in MetNetDB (Locations are based as possible on GO categories)

Table S3. The interaction types defined in MetNetDB

Table S4. Rules of the labeled graph model used in MetNetDB

Figure S1. Gene annotation from multiple sources
Table S1. Data sources of MetNetDB

<table>
<thead>
<tr>
<th>Database</th>
<th>Format</th>
<th>Information retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>AraCyc</td>
<td>Plain text files organized according to frame data model</td>
<td>Pathways, interactions and biomolecules participated in. Name, synonyms, references, comments. Majority metabolic pathways in MetNetDB come from AraCyc</td>
</tr>
<tr>
<td>AGRIS</td>
<td>Plain text files organized according to simple graph model</td>
<td>Transcription network, references and binding sites of individual transcriptional factors</td>
</tr>
<tr>
<td>GO</td>
<td>MySQL dump files organized according to acyclic directed graph data model</td>
<td>The whole copy of gene ontology database</td>
</tr>
<tr>
<td>TAIR</td>
<td>Plain text files (Tabular data)</td>
<td>Affymetrix array elements and their corresponding LocusID mapping, Unitprot ID, TargetP location of polypeptides, loci of each AraCyc pathway</td>
</tr>
<tr>
<td>MapMan</td>
<td>Excel files (Tabular data)</td>
<td>Gene annotation, MapMan BIN ID, gene function category</td>
</tr>
<tr>
<td>BioCyc open chemical compound database</td>
<td>Plain text files organized according to frame data model</td>
<td>UNIQUE-ID, synonyms</td>
</tr>
<tr>
<td>ChEBI</td>
<td>MySQL dump organized according to directed graph data model</td>
<td>ChEBI ID, formula, molecular weight, IUPAC, SMILES</td>
</tr>
<tr>
<td>PubChem</td>
<td>XML files organized according to object data model</td>
<td>PubChem CID, synonyms</td>
</tr>
<tr>
<td>NCI</td>
<td>Structure data format according to object data model</td>
<td>Synonyms, CAS registry number</td>
</tr>
<tr>
<td>KEGG</td>
<td>Plain text files (for compounds) organized according to object data model</td>
<td>Synonyms</td>
</tr>
<tr>
<td>SUBA</td>
<td>Excel file</td>
<td>Protein subcellular location including experiment verified and software predicted</td>
</tr>
<tr>
<td>PPDB</td>
<td>Tabular data</td>
<td>Curated protein subcellular location, especially those in plastid</td>
</tr>
<tr>
<td>AMPDB</td>
<td>Tabular data</td>
<td>Mitochondrion proteins, the subcellular location comes from computational prediction</td>
</tr>
<tr>
<td>AtNoPDB</td>
<td>Tabular data</td>
<td>Nucleolar proteins, subcellular location comes from prediction and experiments</td>
</tr>
<tr>
<td>AraPerox</td>
<td>Plain text</td>
<td>Putative proteins in peroxisomes. Subcellular location comes from literature and computational prediction</td>
</tr>
<tr>
<td>Database</td>
<td>Format</td>
<td>Information retrieved</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>plprote</td>
<td>Plain text files organized</td>
<td>Subcellular location comes from TargetP prediction</td>
</tr>
<tr>
<td></td>
<td>according to object data model</td>
<td></td>
</tr>
<tr>
<td>BRENDA</td>
<td>Plain text files organized</td>
<td>Enzyme’s interaction, substrate, product, activator, inhibitor,</td>
</tr>
<tr>
<td></td>
<td>according to object data model</td>
<td>synonyms, metal ions, references</td>
</tr>
<tr>
<td>MetNet curator</td>
<td>Manually curation</td>
<td>All, with focus on signal transduction information</td>
</tr>
</tbody>
</table>

**Table S2.** The subcellular location hierarchy defined in MetNetDB (Locations are based as possible on GO categories)

**Subcellular location hierarchy**

- plastid
  - plastid envelope
    - plastid outer envelope
      - integral to plastid outer envelope
      - peripheral to plastid outer envelope
      - peripheral to cytosolic side of plastid outer envelope
    - plastid intermembrane space
  - plastid inner envelope
    - integral to plastid inner envelope
    - peripheral to plastid inner envelope
    - peripheral to stromal side of plastid inner envelope
  - plastid stroma
  - plastid inner membrane
  - plastid outer membrane

- thylakoid
  - thylakoid membrane
    - integral to plastid thylakoid membrane
    - peripheral to plastid thylakoid membrane
    - peripheral to stromal side of plastid thylakoid membrane
    - peripheral to luminal side of plastid thylakoid membrane
  - thylakoid lumen
  - thylakoid inner space

- plastoglobules
- plastid nucleoid
- plastid ribosome

- cytosol

- mitochondrion
  - mitochondrial matrix
  - mitochondrial inner membrane
  - mitochondrial outer membrane
  - mitochondrial intermembrane space
  - mitochondrial crista

- apoplast
- autophagosome
- endoplasmic reticulum
  - plasmodesmatal endoplasmic reticulum
Subcellular location hierarchy

- Rough endoplasmic reticulum
  - Rough endoplasmic reticulum cisterna
  - Rough endoplasmic reticulum lumen
- Smooth endoplasmic reticulum
  - Smooth endoplasmic reticulum cisterna
  - Smooth endoplasmic reticulum lumen
- Golgi apparatus
  - GARP complex
  - Golgi lumen
  - Golgi membrane
  - Golgi stack
  - Golgi cis-face
  - Golgi trans face
  - Golgi vesicle
    - ER-Golgi transport vesicle
    - Inter-Golgi transport vesicle
    - Trans-Golgi network transport vesicle
- Inner membrane
- Lipid particle
- Microbody
  - Microbody space
  - Microbody lumen
  - Microbody membrane
- Membrane
- Nuclear membrane
  - Nuclear inner membrane
  - Nuclear outer membrane
- Nuclear lumen
- Nucleolus
- Nuclear body
- Nucleoplasm
- Plasma membrane
- Vacuole
  - Vacuolar lumen
  - Vacuolar membrane
- Not assigned

Table S3. The interaction types defined in MetNetDB

<table>
<thead>
<tr>
<th>Interaction types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enzymatic reaction</td>
</tr>
<tr>
<td>Catalysis</td>
</tr>
<tr>
<td>Translation</td>
</tr>
<tr>
<td>Transcription</td>
</tr>
<tr>
<td>Composition-AND</td>
</tr>
<tr>
<td>Composition-OR</td>
</tr>
<tr>
<td>Diffusion</td>
</tr>
</tbody>
</table>
**Interaction types**

Transport
- Channel-type facilitors
- ATP-driven Transporters
- PEP-dependent Transporters
- Decarboxylation-driven Transporters
- Electron-flow-driven Transporters
- Light-driven Transporters
- Mechanically-driven Transporters

Positive regulation (indirect or unknown mechanism)
- Allosteric activation
- Competitive activation
- Covalent modification
- Complex formation (yielding active protein)
- Transcriptional activation (unknown mechanism)
  - direct
  - coactivation
- Translational activation
- Indirect activation

Negative regulation (indirect or unknown mechanism)
- Allosteric inhibition
- Competitive inhibition
- Covalent modification
- Complex formation (yielding inactive protein)
- Transcriptional inhibition (unknown mechanism)
  - direct
  - corepression
- Translational inhibition
- Indirect inhibition

Degredation

Two-component regulators
- Bind
  - Act as adaptor protein (specific case of binds)

Others (user submitted, curator evaluated)

---

**Table S4. Rules of the labeled graph model used in MetNetDB**

**Rules of the MetNetDB labeled graph model**

- A node must either be a biomolecule or a biochemical reaction
- Genes should be in nucleus, mitochondrion or plastid for plants
- An enzyme must be either a polypeptide or a protein complex
- The substrate and the product of a transportation must be the same biomolecule but in different subcellular locations
- A pathway graph should contain no duplicated nodes which represent same biomolecules in same subcellular location
- A pathway graph should contain no duplicated nodes which represents same biochemical reactions
- An interaction must have at least one substrate and one product
- There must be an interaction node between any two biomolecular nodes in a pathway graph
Rules of the MetNetDB labeled graph model

No two interaction nodes can be adjacent except a catalysis node and an enzymatic reaction node

A pathway graph should contain no orphan nodes which do not connect to any other nodes

A biomolecular node should have a single subcellular location value

Figure S1. Gene annotation from multiple sources. For each locus and its gene product, the annotations from GO, TAIR, MapMan and AGRIS are displayed.

Figure S2. The annotation for the metabolite ethylene. The synonyms of ethylene come from KEGG, PubChem, AraCyc, ChEBI and NCI. The chemical formula, IUPAC ID, SMILES and external database cross-references are from ChEBI. MetNetDB curator tool is used for the visualization.
Acknowledgments

We are very grateful to Seung Y. Rhee, Peifeng Zhang, Lukas Mueller and the AraCyc team for their pathway data. We thank the experts for their assistances in their specific research areas. In particular, we thank Björn Usadel and Mark Stitt, (MapMan gene annotation), Peter Hedden (GA signaling transduction), Basil Nikolau (cuticular wax), Diane Bassham (autophagy), Ron Mittler (oxidative stress), Martha James (starch metabolism), Yingjun Wang (central metabolism), Jianling Peng (several signaling transduction pathways), Pal Narinder (several signaling transduction pathways), and Mentzen Wieslawa (gene correlation data used for the inexact pathway query). Finally, we thank all members of the MetNet group for their valuable input and suggestions. This project is funded by NSF Arabidopsis 2010 #0520267.

References


de Matos P, Ennis M, Darsow M, Guedj M, Degtyarenko K, Apweiler R (2006) ChEBI - Chemical Entities of Biological Interest. NAR Molecular Biology Database Collection entry number 646


CHAPTER 3. A GRAPH QUERY SYSTEM FOR BIOLOGICAL NETWORKS DATABASES BASED ON SUBGRAPH EXTRACTION

A paper to be submitted to *BMC Bioinformatics*

Jie Li, Leslie Miller, Eve Syrkin Wurtele

Abstract

Pathway query is essential in data integration for biological network databases and comparative studies of biological networks. In this paper, we present a pathway query system for biological network databases based on inexact subgraph isomorphism and a novel algorithm of subgraph extraction. The system allows queries based on selected nodes with or without edges among them. In addition, approximate query is supported based on the biomolecular similarity inferred from expression data, gene ontology (GO) and chemical ontology (ChEBI). This pathway query system provides a tool for network comparison. A biologist-friendly visual pathway query system is implemented as the front-end of the algorithm. Its use as a curator tool for the MetNetDB database detailed.

3.1 Introduction

Biological networks provide a context to integrate various types of high-throughput experimental data (genomic, proteomic, metabolic or metabolic flux data) for systematic analysis. Many types of biological network databases have been created in the past dozen years (Lemer et al., 2004; Hashimoto et al., 2005; Karp et al., 2005; Zhang et al., 2005; Ozsoyoglu et al., 2006; Wurtele et al., 2007). These network databases typically contain pathways that can be used as reference sources for computational systems biology. Since the data in biological network databases contains complex relationships among individual data entries, it requires a powerful graph query system other than the
conventional keyword based query interface for understanding, curating and evaluating biological networks in these databases.

First, a graph query system expedites the research in placing experimental data in the context of the biological networks. For instance, a biologist may obtain a list of genes of interest from microarray experiments. These genes might be highly correlated across the experiments. The biologist may be interested in identifying a sub-network from a biological network database that combines genes on this gene list and the interactions among them and other genes. Although biologists can examine graphs representing gene networks manually to obtain this information, this is intractable if the biological network database is huge with respect to screen size and human memory capacity. In addition, high-throughput data requires an automated approach for such types of queries. A single sample of high-throughput data would consist of thousands of genes or protein and hundreds of metabolites for bacteria, up to tens of thousands of genes, proteins or metabolites for higher eukaryotes such as mammals or plants. These biological network databases represent a variety of types of known and/or hypothesized interactions among two or more of several types of biomolecules. For a given species, a graph query system can extract the relevant networks accurately and completely.

Second, a graph query system can be used to compare biological networks across species. Biologists could use the alignment results to identify conserved structures in biological networks, even though a subset of biomolecules and interactions are not identical among species. An alignment created by a graph query does not require that the aligned structures be exactly matched. Rather, some connections among biological entities can be lost for one species. The graph query system can report the number of edge deletions or additions among conserved structures across different species. This information could be used to infer the evolutionary events following diversion of these species.

Third, pathway querying could provide a powerful tool to curate biological network databases. For instance, when a pathway is to be created in a database, curators need to know whether a partial
structure of the pathway is already in the database, so interactions in the new pathway can be created based on the existing structure. Such graph-based queries could prevent redundant structures from being stored in the database.

Finally, a graph query system could be used to integrate hypothetical pathways. Biological interactions computationally inferred from the literature provide a key potential method to expand biological network databases (Dickerson et al., 2003; Yuryev et al., 2006). However, the interaction structures derived by computational inference may be incomplete or unreliable. To help evaluate an interaction or sets of interactions, a graph query system could be used to compare or align them with existing pathways or networks in the database. Curators could then accept or reject the new structures.

The remainder of the paper is organized as follows. First, we introduce the existing graph query tools and algorithms used for biological networks. Second, we briefly present the labeled graph model used in the biological network database, MetNetDB (Wurtele et al., 2003; Wurtele et al., 2007). Based on this labeled graph model, a subgraph extraction method is described in detail. Then, an extension of subgraph extraction to inexact matching is presented. We also explain how the similarity scores are computed for each type of biological entities in MetNetDB. Finally, a working application for MetNetDB is demonstrated.

3.2 Related works

In conventional biological networks, there are several approaches to obtain data of biological interest to a particular researcher from the database. First, most databases organize the pathway data for browsing. For instance, EcoCyc (Keseler et al., 2005) and other BioCyc (Karp et al., 2005) databases organize the pathways based on the functional classes, like “biosynthesis”, or “signal transduction”. Biologists can go through a hierarchically organized arrangement of pathways to identify interesting pathways. Another method of query is based on the keyword search. For instance, both KEGG (Kanehisa et al., 2004) and BioCyc databases provide query interfaces based on names or synonyms
of biomolecules or pathways. More powerful queries for properties of biomolecules and pathways can be created by users if they understand the data model and programming interface of the database. For example, if users are familiar with the frame data model (Karp and Paley, 1996), they can use PerlCyc and JavaCyc (Krishnamurthy et al., 2003) to generate complex queries for BioCyc data.

In addition to the conventional keyword query interfaces, PathCase (Ozsoyoglu et al., 2006), PaVESy (Ludemann et al., 2004), PATIKA (Demir et al., 2002; Nisanci, 2003; Dogrusoz et al., 2006) provides various query interfaces according to the biological network structure. PathCase uses simple graph properties like “find paths between two molecules in a pathway” or “processes within a given number of steps from a process in a pathway”. PaVESy allows users to specify a “start compound” and an “end compound” to create a pathway automatically based on depth-first search and breadth-first search method (Cormen, 2001) if all intermediate biomolecules and interactions have been created and stored in the database. PATIKA can perform queries like “search for pathways containing certain states or transitions” or “search for states in a certain neighborhood of a state or transition”. The NEON component (Nisanci, 2003) of PATIKA even provides a script-based framework for querying. Using this script subsystem, complex queries for pathways can be created based on the data storage model of PATIKA. Although these query features are more powerful than the keyword searching, they either are limited by the predefined query statements or require users have knowledge of internal data organization and script programming, which are difficult for most biologists.

Several graph query tools have been created for biological network databases. However, these have been designed for special purposes, and have limitations for large-scale graph queries. For instance, NetMatch (Ferro et al., 2007) is a useful Cytoscape plugin (Shannon et al., 2003) for querying motifs with unknown path length but it can only support queries based on exact matches of node attributes. Tohsato (Tohsato et al., 2000) presents a multiple alignment method that can only be applied to simple linear structures; however, biological networks tend not to be linear. MetaPathwayHunter (Pinter et al., 2005) implements a tree isomorphism algorithm, that can be
applied to biological networks with tree structures, which are more complex than a simple linear structure. However, the tree isomorphism algorithm would require manually removing cycles if a network contains loops. Such a requirement makes it impractical for large-scale application, since loops are very common and functionally important structures in biological networks (Milo et al., 2002; Barabasi and Oltvai, 2004). SAGA (Tian et al., 2007) implements an index-based clique finding approach to solve an inexact subgraph isomorphism problem. SAGA first connects each possible matching node pair in the two graphs with an edge. Thus, the isomorphism problem is converted to a problem to find a maximum clique in this new graph. SAGA can be applied to networks with various structures, including cycles. It also supports inexact graph matching if the scores of node mismatches are defined. However, it does not consider the edge penalty for individual edges. Rather, the definition of structure distance depends only on the shortest path between node pairs. Because of the limitation of the algorithm, it cannot support querying high-density graphs. Finally, SAGA does not distinguish various types of gap nodes. Therefore, SAGA does not always return the biologically optimal matching results, as described in Figure 1. In our MetNet platform (Wurtele et al., 2003; Wurtele et al., 2007), FCModeler (Dickerson et al., 2003), now implemented as plugins of Cytoscape, is a modeling tool with several graph analysis features. It can visualize and animate gene expression data in a biological network. It also can find paths and cycles in a pathway, and extract subgraphs according to \( p \)-neighborhood or pathway names. However, it cannot extract all paths between all pairs of selected biomolecules in the biological network database. Suppose biologists select two genes in a pathway, if the distance between the two genes in the pathway is large, the \( p \)-neighborhood method provided by FCModeler covers many biomolecules that may be relevant to only one of the genes since the \( p \)-neighborhood does not consider if a neighbor is on the path toward the other gene.

Messmer’s subgraph isomorphism algorithm (Messmer and Bunke, 1993; Messmer and Bunke, 1998) allows inexact subgraph matching between an input graph and a graph database. In addition, the algorithm can improve query performance by preprocessing member graphs in the graph database.
However, similar to SAGA, Messmer's algorithm, as well as all subgraph isomorphism algorithms, does not guarantee to provide a biologically optimal matching.

Figure 1 Common substructure and subgraph isomorphism. Graph a is an input query graph. Graph b is the plastidial part of Acetyl-CoA biotin network (Fatland et al., 2005). Subgraph c is the best matching part in b for a. Graph d is the cytosolic part of Acetyl-CoA biotin network. Subgraph e is the best matching part in d for a. Now we denote the matching cost between the graph a and b as $C(a, b)$. Similarly, we have $C(a, c)$, $C(a, d)$, and $C(a, e)$. Obviously $C(a, e) < C(a, c)$. However, there are a larger part of graph d than the one of the graph c that is not matched. Therefore, for some setting of graph operation costs, $C(a, d) > C(a, b)$ because the additional deletion cost is included in the subgraph isomorphism algorithm. This means, the subgraph isomorphism is not always returns the best matching from the biological perspective.

Figure 1 shows that why subgraph isomorphism algorithms including SAGA and Messmer’s algorithm cannot get the biologically optimal alignments between the input graph and graphs in a
database. The input query graph $a$ exactly matches a subgraph $e$ of the graph $d$, which is the cytosolic part of Acetyl-CoA biotin network (Fatland et al., 2005). The graph $a$ approximately matches the subgraph $c$ of the member graph $b$, which is the plastidial part of Acetyl-CoA biotin network. From the biological perspective, the matching cost $C(a, e) < C(a, c)$, and thus graph $d$ should be picked as a better match than the graph $b$. However, because of the additional node deletion, in the subgraph isomorphism algorithm, we have $C(a, d) > C(a, b)$, which means the graph $b$ is picked as the better match. This means, the subgraph isomorphism is not always returns the best matching from the biological perspective.

To address this issue, we have to get the common substructures between two graphs. Our solution is extracting the substructure from the alignment results of the subgraph isomorphism. In Figure 1, once the alignment between graph $a$ and $b$, and the alignment between graph $a$ and $d$ are obtained, the subgraph $c$ of $b$ and subgraph $e$ of $d$ are extracted based on the matched nodes. Therefore, the matching scores of $(a \leftrightarrow c)$ and $(a \leftrightarrow e)$ can be calculated. Finally, the alignment results can be sorted based the matching scores.

Interestingly, there are only a few studies addressing the problem of extracting “biological relevant” sub-networks based on selected input nodes. The definition of “biological relevant” depends on the application of those algorithms. One method uses “commute distance” to extract an interesting subgraph (Sébastien et al., 2005). However, the commute distance is based on the probability of edges in a random walk on the graph. This static analysis of a graph cannot reflect the dynamic behavior of biological networks. Some steps in a biological network may not be activated initially. Later these steps will be activated if some biochemical signals are present. The commute-distance approach may lose such structures since the probability can only be assigned by a fixed value before the random walk. In addition, the relevance between the edge probability in a random walk and the reaction properties in the biological networks is unclear.
Another method uses a significant area search to collect subgraph from a list of selected nodes (Sohler et al., 2004). Significant area is a subgraph that meets the predefined significant criteria. First, they infer the possibility of edge extension from one node based on the functional relationship between proteins in literature networks or co-regulation pathways in metabolic networks. The expanded regions are combined and the overlapped parts are pruned. This method is same as the \( p \)-neighborhood expansion, except that the expansion possibility is determined by a significance value inferred from the functional relationship among nodes. Thus, this method is inefficient if the distances between pairs of selected nodes are large.

We developed the subgraph extraction algorithm, which complements the subgraph isomorphism algorithm, to identify the common substructure between the two graphs. The subgraph extraction algorithm is designed to extract the complete context of the selected nodes. Thus, it returns all paths among the selected nodes. All alignments between the input graph and the member graphs in the database are evaluated based on extracted common substructures. In addition, the subgraph extraction algorithm itself can also be used to identify interesting sub-networks based on selected biomolecules without edges. Thus, biologists can map the experimental data onto the biological networks in the database. For example, In the MetNet platform, biologists can use exploRase (Wurtele et al., 2003; Lee et al., 2004; Wurtele et al., 2007) to statistically and visually analyze a set of experimental data and from these analyses manually select a list of biomolecules based on selected criteria about their changes in accumulation. These “interesting” biomolecules can be used to extract subgraphs from the biological network database. Then, the function of these genes or metabolites can be studied in further pathway analysis.

Two types of queries have been developed for subgraph extraction in MetNetDB. One is an exact match query, which means the labels of the input nodes must be matched exactly with the labels of nodes in the member graph. The other is an inexact match query, which allows approximate matching
between input nodes and graph nodes, with substitution costs. The label matching and substitution can be performed on all attribute-value tuples or partial tuples of the corresponding labels.

### 3.3 The MetNetDB storage model

<table>
<thead>
<tr>
<th>Terminology or symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injection or injective function</td>
<td>A function $f$ maps a distinct $x$ in set $X$ to a distinct $y$ in set $Y$</td>
</tr>
<tr>
<td>Bijection or bijective function</td>
<td>A function $f$ such that for every $y$ in set $Y$, there is exactly one $x$ in $X$ such that $f(x) = y$</td>
</tr>
<tr>
<td>Clique</td>
<td>An undirected graph such that there is an edge between every two nodes in this graph</td>
</tr>
<tr>
<td>Induced subgraph</td>
<td>A subgraph $G_S$ of a graph $G$ is said to be induced by $S$ if $S$ is a subset of nodes in $G$, and for any pair of vertex $x$ and $y$ in $S$, $e = (x, y)$ is an edge of $G_S$ if and only if $e$ is an edge of $G$</td>
</tr>
<tr>
<td>Association graph</td>
<td>Given two node sets $X$, $Y$ and association rules $R$ and co-occurrence rules $C$, an association graph $G$ is a graph such that if $x$ in $X$ and $y$ in $Y$ satisfies the association rule $R$, there is a node $v = x \circ y$ in $G$, and if $v_1$ and $v_2$ in the $G$ satisfy the rule $C$, there is an edge $e = (v_1, v_2)$ in $G$</td>
</tr>
<tr>
<td>Input node and input graph</td>
<td>The nodes input by users for queries are input nodes. In subgraph extraction, the input nodes define the boundary of the subgraph. If uses input a graph for querying, this graph is called input graph</td>
</tr>
<tr>
<td>System node and System graph</td>
<td>A system graph $G_S = (V_S, E_S)$ is an induced subgraph of $G = (V, E)$ defined by $S \subseteq V$, in which any $v \in V_S$ if and only if $v \in S$ or $v$ is on the path of $(v_1, v_2)$ such that $v_1 \in S$ and $v_2 \in S$. Here $S$ is the input node set specified by users. $V_S$ is the system node set</td>
</tr>
<tr>
<td>Graph databases</td>
<td>A collection of graphs</td>
</tr>
<tr>
<td>Member graph</td>
<td>Member graph is the individual graph in a graph database. Since these graphs are already known, they can be processed in advance to improve performance of graph queries. This concept is same as “model graph” in Messmer’s subgraph isomorphism algorithm</td>
</tr>
<tr>
<td>Similarity and distance</td>
<td>Both similarity and distance define the degree of resemblance between two objects. In this paper, similarity has a range $[0, 1]$ in which 1 means that the two objects are identical. Distance has a range $[0, +\infty)$ in which 0 represents that the two objects are identical</td>
</tr>
</tbody>
</table>
| Big-O notation: $O(\bullet)$, $\Omega(\bullet)$ and $\Theta(\bullet)$ | Big-O notation is used to describe the asymptotic property of functions. They are used in the analysis of the complexity of algorithm (Cormen, 2001). For instance, $f(n) \in O(g(n))$ means $g(n)$ is an asymptotic tight bound of $f(n)$. We say $f(n) \in O(g(n))$ as $n \rightarrow +\infty$ if and only if exists $n_0$, $M > 0$ such that $|f(n)| \leq M|g(n)|$ for any $n > n_0$. $f(n) \in \Omega(g(n))$ as $n \rightarrow +\infty$ if and only if exists $n_0$, $M > 0$ such that $|f(n)| \geq M|g(n)|$ for any $n > n_0$. $f(n) \in \Theta(g(n))$ if
<table>
<thead>
<tr>
<th>Terminology or symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biomolecules</td>
<td>Biomolecules are physical entities that participate in biochemical interactions. They can be proteins, RNAs, genes, metabolites or other chemical substances.</td>
</tr>
<tr>
<td>Entity with location</td>
<td>An biomolecule in a specific subcellular location</td>
</tr>
<tr>
<td>Interaction</td>
<td>An interaction is a biochemical event or a relationship amongst entitywithlocations</td>
</tr>
<tr>
<td>Pathway</td>
<td>A pathway is a collection of interactions, which are grouped by biologists according to a function, such as TCA cycle, fatty acid synthesis, GA signaling etc.</td>
</tr>
</tbody>
</table>

MetNetDB (Wurtele et al., 2003; Wurtele et al., 2007) storage model represents biological networks as labeled graph model. The current focus of this database is the model plant species, Arabidopsis. MetNetDB is organized into pathways. These pathways frequently overlap with each other and with new pathways from biologists and other data sources. Each pathway is comprised of several interactions. Each interaction affects one or more biomolecules. MetNetDB incorporates annotation from other databases, and from biologists, who are experts in the area to which they contribute. These annotations are associated with pathways, interactions and entities.

The labeled graph model designed for MetNetDB is based on directed graphs. This labeled graph model is a tuple $G = (V, E, f_V, f_E, L, O, R)$ in which $V$ is the node set in the graph, $E$ is the edge set, $L$ is the total label set of the graph, $f_V$ is a node label assignment function $f_V : V \rightarrow L$. $f_E$ is an edge label assignment function $f_E : E \rightarrow L$. $O$ is allowed operations on the graph model. The operations can be: node insertion, node deletion, label substitution of nodes and edges, edge insertion, and edge deletion. These five edit operations allow us to change one graph to another graph. $R$ is the rule of the graph model for data integrity, which is either obtained from biological knowledge, or from database and graph requirements. Rules can be described as a set of Boolean functions mapping from nodes, edges, and labels to a Boolean domain like $r : V \times E \times L \rightarrow \{0,1\}$.
Figure 2 A labeled graph model representation for interactions. There are five entity type nodes (circle shape) and two interaction type nodes (rectangle shape). The labels of node n₁ and n₄ are shown in the table. The names are also shown in all nodes. The edge label “2” of the edges between node n₃ and n₄, and node n₄ and n₇, are coefficients of the corresponding biological entities H⁺ and CO₂.

To represent biological networks in this labeled graph model, we use the following mapping approach (Figure 2). Nodes correspond to biomolecules and interactions in the networks. Edges represent the relationships among biomolecules and interactions. For example, if a biomolecule is a causal factor of a regulatory interaction, there is an edge from the biomolecule to the interaction. On the other hand, if a gene is affected by a regulatory interaction, there is an edge from the interaction to the gene. Edges can also be used to connect two nodes representing interactions in MetNetDB if one interaction is “catalysis” and the other is the corresponding “enzymatic interaction”. If a node represents a biomolecule, the node label in the MetNetDB graph model can include the biomolecular name, synonyms, organism, subcellular location, entity type, references, user comments, location confidence, function confidence, and annotations from external databases such as gene functions or chemical properties of metabolites. If a node represents an interaction, the node label can include interaction name, synonyms, interaction type, confidence, EC number, references, and comments. The edge label stores coefficients (how many of each biomolecule participate in an interaction) or
turnover numbers (how many moles of a substrate that an enzyme can convert to product per catalytic site per unit time). Thus, the whole network is represented as a labeled graph model.

3.4 Definition of subgraph extraction

Subgraph extraction means that the query is a subgraph based on a list of input nodes without any topological relation defined among them. In the following section, we use “system” to describe interesting subgraphs (Figure 3). More strictly, a system subgraph $G_S = (V_S, E_S)$, where $G_S$ is a system subgraph, $V_S$ are the nodes in $G_S$ and $E_S$ are the edges in $G_S$, is an induced subgraph of $G = (V, E)$ defined by $S \subseteq V$, in which any $v \in V_S$ if and only if $v \in S$ or $v$ is on any path of $(v_1, v_2)$ such that $v_1 \in S$ and $v_2 \in S$. In the remaining part of the paper, we refer to $S$ as input nodes, $V_S$ as system nodes. In the graph shown in Figure 3, $S = \{n_3, n_5, n_{15}\}$ and $V_S = \{n_3, n_4, n_5, n_7, n_8, n_9, n_{10}, n_{11}, n_{14}, n_{15}, n_{16}, n_{17}\}$. The subgraph circled by the dotted line is the system graph $G_S$. Finally, the question is that based on a node set $S$ and a graph $G$, how we can find the system graph $G_S$. In contrast to a subgraph isomorphism, this query type only relies on the input nodes themselves; it does not need any edge information among these input nodes.

Figure 3 The definition of a system subgraph. In this graph, the circled nodes (n3, n5, n15) are the input query nodes. The subgraph within the dotted line part is a system defined by these input nodes.
3.5 Exact subgraph extraction algorithm

Exact subgraph extraction means the input nodes are always specified from the nodes in the graph to be queried. In contrast, inexact subgraph extraction means the input nodes are not necessarily in the graph to be queried. In this section, we first introduce the algorithm we have developed for exact subgraph extraction. The basic idea of this algorithm is to identify all nodes on all possible paths $V_s$ among the input query node set $S$. Then, we use these nodes to induce the system graph. By this way, we do not need to identify all possible paths in the system graph. Since cycles are very common in biological networks, this approach can avoid computationally expensive steps of identifying all possible paths among the input nodes. In the following sections and figures, the input node set $S$ has circle shape. There are two steps to identify the system nodes that are not in $S$. The system nodes identified in the first selection step has hexagon shape, and those identified in the second selection step has square shape. The nodes marked by circle, hexagon and square shapes can be proved as the complete system node set. Therefore, we can induce the system graph by these nodes.

3.5.1 Algorithm

The exact subgraph extraction algorithm contains seven steps. In addition, there is a preprocessing step and a post-processing step.

Preprocessing

Our algorithm works on an undirected connected graph, while many biological networks are modeled as directed graphs, except for protein-protein interaction networks. Therefore, the input graph to be queried is converted to an undirected connected graph, if necessary. The edge direction can be saved for later recovery. Likewise, disconnected graphs are converted to connected graphs by identifying the connection components of the input graph and applying the algorithm to each component separately. In MetNetDB, the disconnected graphs usually come from those superpathways (e.g.
superpathway of fatty acid biosynthesis) that contain several functionally relevant pathways. In the post-processing step, the separated interesting subgraphs are combined.

**Generate a spanning tree from the undirected graph**

A spanning tree $T$ is a subgraph of $G = (V,E)$ that contains all nodes of $V$. A depth first search (DFS) (Cormen, 2001) is used to create a search tree of the input graph. It does not matter that which node is selected as the starting node for DFS or how the DFS is applied to the input graph. The search tree is therefore a spanning tree of the input graph. A node is randomly picked as a root node of the tree and all edges are assigned with the directions away from the root node. The spanning tree is used to find a path between any pair of nodes in the graph efficiently.

**Efficiently find a path between any pair of nodes in the spanning tree**

![Diagram](image)

Figure 4 Efficient path finding. This figure shows three cases of finding path between node pairs $n_1$ and $n_2$. In (a), the two paths are joined at the root node. In (b), the two paths are joined at some intermediate nodes. In (c), the two paths are joined at either $n_1$ or $n_2$. $n_R$ is the root node of the spanning tree. $n_{M}$ in (b) is the intermediate node at which path $p_{1\rightarrow R}$ and $p_{2\rightarrow R}$ are joined, where $p_{1\rightarrow R}$ is from $n_1$ to $n_R$ and $p_{2\rightarrow R}$ is from $n_2$ to $n_R$.

To find a path between any pair of nodes in the spanning tree efficiently, we traverse from both nodes to the root node of the spanning tree. The two traversal paths may be joined in three cases (Figure 4).
a) The two paths are joined at the root node. Then the path between two nodes is the combination of the two paths. For instance, in Figure 4(a), path \( p_{1\rightarrow R} = s_1 \), and path \( p_{2\rightarrow R} = s_2 \). Then, path \( p_{1\leftrightarrow 2} = s_1 + s_2 \).

b) The two paths are joined at an internal node that is neither the root node nor either of the two starting nodes. Then, the two paths share a common path to the root node. The node set on the path between the two nodes is the difference between the node sets on the two paths. For instance, in Figure 4(b), path \( p_{1\rightarrow R} = s_1 + s_3 \), and path \( p_{2\rightarrow R} = s_2 + s_3 \). Then, path \( p_{1\leftrightarrow 2} = s_1 + s_2 \).

c) The two paths are joined at one of the starting nodes. Then, the two paths share a common path to the root node. The node set on the path between the two nodes is the difference between the node sets on the two paths. For instance, in Figure 4(c), path \( p_{1\rightarrow R} = s_1 \), and path \( p_{2\rightarrow R} = s_1 + s_2 \). Then, path \( p_{1\leftrightarrow 2} = s_2 \).

The node where the two paths are joined can be found by Tarjan’s offline least common ancestors algorithm (Cormen, 2001).

Mark input nodes and nodes on the path between any pair of input nodes

The input nodes \( S \) are marked with circle to indicate that these nodes are on the boundary of the system graph. The nodes on the path are marked by hexagon shape to indicate these nodes belong to system nodes \( V_S \) of the system graph \( G_S \) but not on the boundary \( S \) of the system. In Figure 5, \( V_{\text{CIRCLE}} = \{n_3, n_5, n_{15}\} \), and \( V_{\text{HEXAGON}} = \{n_4, n_7, n_8, n_9, n_{10}, n_{14}\} \).

Identify cycles in the graph

Since the spanning tree \( T = (V, E_T) \) is an acyclic subgraph of the original graph \( G = (V, E) \), thus the edge number of the tree is \( |E_T| = |V| - 1 \). So the number of edges \( e \) that satisfies \( e \in (E - E_T) \) is
Adding one of the edges in $E - E_T$ to the spanning tree $T$ will generate a cycle $C$ in the graph $G$. By this way, we will get $|E| - |V'| + 1$ cycles of the graph $G$. In Figure 6, three cycles ($C_1 = \{n_1, n_2, n_3\}$, $C_2 = \{n_8, n_9, n_{10}, n_{11}\}$ and $C_3 = \{n_{14}, n_{15}, n_{16}, n_{17}\}$) exist.

Figure 5 The first selection step to identify the system nodes. The path $(n_3, n_4, n_5)$, $(n_3, n_4, n_7, n_8, n_9, n_{10}, n_{14}, n_{15})$ and $(n_{15}, n_4, n_7, n_8, n_9, n_{10}, n_{14}, n_{15})$ are paths between all pairs of input query nodes $(n_3, n_5), (n_3, n_{15}), (n_5, n_{15})$. These paths are represented by double line. All nodes on these paths are selected by the algorithm as system nodes. The input query nodes $(n_3, n_5, n_{15})$, have a circle shape, and other system nodes $(n_4, n_7, n_8, n_9, n_{10}, n_{14})$ are hexagons.

Create connected cycle sets

For each pair of cycles, the intersection of nodes is enumerated and the connection between the two cycles is determined. Finally, connected cycle sets are created by grouping these connected cycles. In our implementation, we create an undirected graph $G'$ in which the nodes are nodes of original graph $G$ and the cycles identified in the step “identify cycles in the graph”. If one cycle contains one node in $G$, then there is an edge between the nodes representing the cycle and the node in $G'$. The connected components in $G'$ are used to identify connected cycle sets. In Figure 6, $C_1$, $C_2$ and $C_3$ are disjoint. So the connected cycle sets are $S_1 = \{C_1\}$, $S_2 = \{C_2\}$ and $S_3 = \{C_3\}$.
Figure 6 Cycle identification. The input query nodes are \{n_5, n_7, n_{15}\}. The cycles \(C_1 = \{n_1, n_2, n_3\}\), \(C_2 = \{n_8, n_9, n_{10}, n_{11}\}\) and \(C_3 = \{n_{14}, n_{15}, n_{16}, n_{17}\}\) are identified by a spanning tree method. These cycles are disjoint. The additional system nodes that have been selected in each cycle are indicated by hexagons.

**Mark all nodes in the connected cycle sets based on the existing marked node set**

The marked nodes in each connected cycle set are enumerated. Let the number of circle nodes in a connected cycle set be \(r\), the number of hexagon nodes be \(y\). If \(y + r \geq 2\), then each nodes that has not been marked in this connected cycle set is marked by a square shape. If \(y = 1\) and \(r = 0\), then each nodes that has not been marked in this connected cycle set is marked by a square shape too. These cycles are connected with the main path through one node only. If these cycles are not allowed to be included in the system graph, we can ignore all conditions such that \(y + r \leq 1\). If \(r = 1\) and \(y = 0\), then the connected cycle set is outside the system except the input node which has circle shape, then no more nodes are marked. In Figure 7, node \(n_{11}\) in \(C_2\) and node \(n_{16}, n_{17}\) in \(C_3\) are marked with square shapes. \(C_1\) has only one circle node, so no more nodes are marked.
Figure 7 The second selection step for system nodes. Based on the marking algorithm described in 0, the nodes \( n_{11} \) in \( C_2 \) and \( \{n_{16}, n_{17}\} \) in \( C_3 \) are marked by square shapes. The cycle \( C_1 \) contains only a single circle node (an input query node), which means the cycle is at the boundary of the system and no more nodes need to be added to the system subgraph, \( G_S \).

**Output the subgraph induced by marked nodes**

The subgraph induced by all nodes with marked nodes is returned. In Figure 8, the induced graph is circled by dotted lines.
Figure 8 The induced subgraph is displayed within the dotted line. To generate this subgraph all marked nodes are collected and the subgraph is induced by these nodes. We claim that this subgraph is the system graph according to the definition of system graph shown in Section 3.4.

**Post-processing**

The post-processing restores the edge directions that are saved in the pre-processing step. In addition, if the original graph is a disconnected graph all system subgraphs returned from each connected components are combined.

Although we get the subgraph satisfying the definition of system graph, it may not be complete in the biological context if for an interaction node, some participated biomolecular nodes are included in the system graph and others are not. Therefore, the node labels in the system graph are checked. If the node is an interaction node, all adjacent biomolecular nodes are included in the result graph. If the node is an enzymatic interaction node and it is connected to a catalysis interaction node, the corresponding enzyme node and the catalysis interaction node are included in the system graph too.

**3.5.2 Analysis of the running time of the algorithm**

The subgraph extraction contains several steps. Generating a spanning tree from the graph $G = (V, E)$ requires $O(E + V)$ time. Finding the least common ancestors in the tree for all predefined node pairs
needs $O(V)$ time. Finding any path in the spanning tree needs $O(V)$ time. The first selection step identifies one path for every pair of system nodes. Suppose the number of system nodes is $k$, thus, the first selection step needs $O(k^2V)$. The number of cycles found by the algorithm is $|E| - |V| + 1$. Each cycle has at most $V$ nodes. Thus, the creation of connected cycle sets needs $O((E - V + 1)V)$. The number of connected cycle sets is at most $|E| - |V| + 1$, therefore, the second selection step needs $O(|E| - |V| + 1)$ steps. Thus the overall running time of the algorithm is $O(k^2V + (E - V + 1)V)$. Since $|E| < |V|^2$ and $k \leq |V|$, we have the running time as $O(V^3)$.

The running time $O(V^3)$ means the performance of the subgraph extraction for a biological network data depends on the total number of biomolecules and interactions in a pathway represented by the MetNetDB labeled graph model. This running time is the polynomial function of the node number and can be regarded as an efficient algorithm.

### 3.5.3 Correctness proof of the subgraph extraction algorithm

In this section, we use symbols defined in Table 2 in the proof.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_A$</td>
<td>The edges of cycle sets generated according to the cycle identification algorithm in the step “identify cycles in the graph” for the graph $G$</td>
</tr>
<tr>
<td>$E_G$</td>
<td>All edges of cycles in the graph $G$</td>
</tr>
<tr>
<td>$E_T$</td>
<td>The edges of the spanning tree $T$ created according to the algorithm in the step “generate a spanning tree from the undirected graph”</td>
</tr>
<tr>
<td>$E_U$</td>
<td>All edges of the graph $G$ that are not in $T$</td>
</tr>
<tr>
<td>$C_T$</td>
<td>All edges in a cycle $C$ that belong to $E_T$</td>
</tr>
<tr>
<td>$C_U$</td>
<td>All edges in a cycle $C$ that do not belong to $E_T$</td>
</tr>
<tr>
<td>Minus operator between two cycles: $C_1 - C_2$</td>
<td>Minus means removing all edges in $C_2$ except that shared with cycle $C_1$. The remaining part is still a cycle</td>
</tr>
<tr>
<td>$V_S$</td>
<td>The system subgraph node set defined in Section 3.4</td>
</tr>
<tr>
<td>$V_A$</td>
<td>The node set returned by the exact subgraph extraction algorithm described in Section 3.5.1. This needs to be proved to same as $V_S$, which is the system subgraph node set</td>
</tr>
</tbody>
</table>
Lemma 1. The cycle sets generated by the cycle identification algorithm described in the step “identify cycles in the graph” cover all edges in the graph that belongs to any cycle in the graph.

Proof. We need to prove that $E_G = E_A$. Because it is trivial that $E_A \subseteq E_G$, thus, if we can prove $E_G \subseteq E_A$, then we have $E_A = E_G$.

Now we prove $E_G \subseteq E_A$ by contradiction. Suppose there is a cycle $C \subseteq E_G$, which contains at least one edge $e \in C$ such that $e \notin E_A$. Since the cycle $C$ contains two parts of edges, in which $C_T \subseteq E_T$, and $C_U \subseteq E_U$. Now we pick one edge $e_1 \in C_U$. Since this edge defines a cycle $C_1$, and $C_1 \neq C$ because $C_1$ is a cycle generated by the algorithm described in the step “identify cycles in the graph” and $C$ is a cycle cannot be generated by the same algorithm, we can remove all edges in $C_1$ except those edges shared by $C$. Since $C_1$ and $C$ share these edges, the remaining part $C - C_1$ is still a cycle. Repeating the steps for all edges $e_i \in C_U$, we finally remove all edges in $C_U \subseteq E_U$ and still get a cycle $C_k$. We have $C_k \subseteq E_T$. The fact that $C_k$ is a cycle contradicts the statement that $E_T$ is an edge set of a spanning tree. Therefore, we have $e \in E_G \Rightarrow e \in E_A$, or $E_G \subseteq E_A$. This completes the proof.

Lemma 2 Denote the nodes in the system graph as $V_S$, and the node set of the graph returned by the algorithm is $V_A$. The algorithm described in Section 3.5.1 marks all nodes in the system graph, or $V_S \subseteq V_A$.

Proof. Let $S$ be the user-specified input nodes. We prove $V_S - S \subseteq V_A - S$ by contradiction. Suppose there is a node $v$ such that $v \in V_S - S$ and $v \notin V_A - S$, which means the node belongs to one path $p_1: v_1, \ldots, v_2$ in which $v_1 \in S$ and $v_2 \in S$ but has no been marked. Thus, $v$ is not in another path $p_2: v_1, \ldots, v_2$ in which all intermediate nodes are marked (either hexagon or square) in the algorithm. Therefore, $p_1$ and $p_2$ form a cycle $C$ of which one node $v$ on this cycle has not been marked. This cycle has at least two nodes $v_1$ and $v_2$ that have been marked. However, based on Lemma 1, we have
proved that all edges (thus, all nodes) in a cycle of the graph have been identified. Thus, all nodes including \( v \) in this cycle \( C \) must have been marked. This contradiction means \( V_S - S \subseteq V_A - S \), and completes the proof.

**Lemma 3** Denote the nodes in the system graph as \( V_S \), and the node set of the graph returned by the algorithm is \( V_A \). All nodes marked by the algorithm Section 3.5.1 are nodes in the system graph, or \( V_A \subseteq V_S \).

**Proof.** Let \( S \) be the user-specified input nodes. We prove \( V_A - S \subseteq V_S - S \). All nodes in \( V_A - S \) are marked, either in the first selection step or in the second selection step. If a node \( v \) is marked with hexagon in the first selection step, it is on a path \( p : v_1, \ldots, v_2 \) such that \( v_1 \) and \( v_2 \) have a circle shape, or they belong to \( S \). By definition of a system graph, we have \( v \in V_S \). If \( v \) is in a cycle set \( C \) and is marked with a square shape in the second selection step, then \( v \) is on a path that has two nodes \( x_1, x_2 \) that are marked with shapes (either circle or hexagon) in the first selection step. Thus, there is a path \( p : v_1, \ldots, x_1, D, x_2, \ldots, v_2 \), in which \( D \) is any sub-path from \( x_1 \) to \( x_2 \). Replace \( D \) with the cycle set \( C \), and we can form a different path from \( v_1 \) to \( v_2 \) which contains \( v \in C \), this means \( v \) is on a path \( p : v_1, \ldots, v_2 \) such that \( v_1 \) and \( v_2 \) are in \( S \). Therefore, we have \( v \in V_S \) for all nodes that are marked with square shape in the second step. Combined, we have \( V_A - S \subseteq V_S - S \). This completes the proof.

**Theorem 1.** The algorithm described in Section 3.5.1 can correctly identify the system graph.

**Proof.** Based on Lemma 2 and Lemma 3, we have \( V_S \subseteq V_A \) and \( V_A \subseteq V_S \). Hence, \( V_S = V_A \). This means our algorithm correctly identifies all nodes in the system graph based on the definition of system graph. The proof is completed.

### 3.6 Extension to approximate subgraph extraction

The exact subgraph extraction algorithm requires that all user-specified nodes can be found in the graph. However, biologists often identify biomolecules from experimental data that are yet
incorporated into biological networks. The subgraph extraction based on nodes not in the graph is called approximate subgraph extraction. To use the exact subgraph extraction algorithm for the approximate one, we may substitute these biomolecules with similar biomolecules that are take part in pathways. The substitution should be based on the similarity between biomolecules. Meanwhile, since the compactness of the resultant subgraph of pathways provides hints as to the functional relevance of biomolecules, both the similarity and network compactness should be considered in the substitution.

Next, we introduce the algorithm of node substitution.

3.6.1 Algorithm

The idea is as follows. First, all candidate nodes for each user-specified node are obtained by some similarity measurements outside of this algorithm. We will introduce the similarity measurements used in MetNetDB in Section 3.6.3. Second, the distance between each pair of candidates is computed. Since the candidate nodes are in the graph, we can perform this step in \( \Theta(V^3) \) time by using Floyd-Warshall algorithm (Cormen, 2001). Third, a new graph is constructed based on similarity scores and shortest path distances. Finally, Prim algorithm (Cormen, 2001) is applied to the new graph to get the minimal spanning tree. The nodes in this minimal spanning tree are used as the input nodes \( S \) used to identify the system graph.

Let the graph to be queried is \( G_M \), user-specified input nodes as \( v_1, v_2, \ldots, v_n \), and the new graph constructed by the algorithm is \( G_W \); the candidate nodes in \( G_M \) for \( v_i \) are \( v_i^1, v_i^2, \ldots, v_i^k_i \); the candidates node in \( G_M \) for \( v_j \) is \( v_j^1, v_j^2, \ldots, v_j^k_j \). Now we connect each pair of nodes \( v_x^i \) and \( v_y^j \) in \( G_W \).

The edge weight of \( (v_x^i, v_y^j) \) is \( w(v_x^i, v_y^j) = f(d(v_i, v_x^i), d(v_j, v_y^j), d(v_x^i, v_y^j)) \), where \( d(v_i, v_x^i) \) is a distance function obtained from the similarity measurement between \( v_i \) and \( v_x^i \); \( d(v_j, v_y^j) \) is a distance function derived from the similarity measurement between \( v_j \) and \( v_y^j \). The similarity measurement is domain-
specific. For example, it can be inferred from correlation data or ontology data. $d(v_x^i, v_y^j)$ is the shortest path distance between $v_x^i$ and $v_y^j$ in the graph. The weight is a function of all of three distances ($d(v_i, v_x^i), d(v_j, v_y^j)$ and $d(v_x^i, v_y^j)$).

Let the candidate nodes similar to one user-specified node $v_i$ based on all three distance functions be node group $V_i$. Because each pair of nodes in $V_i$ is similar to each other, we connect each pair of nodes in $V_i$ with a small edge weight $\epsilon$. The weight should be small enough so that the edges among nodes of one candidate group are always in the minimal spanning tree (i.e. the edge is always selected by Prim algorithm). The small edge weight $\epsilon$ can be calculated by this method. Suppose the minimal weight of all pairs of candidate nodes is $\min w(v_x^i, v_y^j) = p$. The maximal number of candidate nodes in any group is $q$. Thus, we can assign the value $\epsilon \leq (p/q)$ if we connect these candidate nodes within one group with a straight line. We could also assign $\epsilon$ as less than $p$ if all candidate nodes in one group are fully connected, but this method increases the edge number of the graph.

The optimal candidate nodes are obtained by extracting a minimal spanning tree from the graph. These resultant candidate nodes are used to find the system graph by using the exact subgraph extraction algorithm of Section 3.5.1.

### 3.6.2 Example

In this section, we show an example of approximate subgraph extraction. In Figure 9, we are trying to use three input nodes $n_a, n_b$ and $n_c$ to extract a subgraph from the graph $G_M$. Table 3 lists the distance among nodes in $G_M$. For instance, in $G_M$, the shortest path between node $n_2$ and node $n_6$ has length 4. The input nodes do not exist in $G_M$ but there are some similar nodes in $G_M$, which are specified in Table 4. For instance, in $G_M$, node $n_2, n_3$, and $n_4$ are similar to the input node $n_a$. 
Figure 9 Example of approximate subgraph extraction. Input node $n_a$, $n_b$, and $n_c$ are nodes used to query. $G_M$ is the graph to be queried. The nodes similar to input nodes are listed in Table 4.

The distances between the input nodes and the nodes in $G_M$ are also listed in Table 4. For instance, $d(n_a, n_2) = 0.10$. If there is not a distance listed for two nodes, this means the distance is infinite. For instance, $d(n_a, n_3) = \infty$.

In this example, we define the edge weight as

$$w(v_i^*, v_j^*) = \frac{f(d(v_i^*, v_j^*), d(v_i^*, v_j^*), d(v_i^*, v_j^*))}{d(v_i^*, v_j^*) + 10d(v_j, v_j^*) + 10d(v_i^*, v_j^*)}$$

Table 5 displays the edge weight as regards to the input nodes. "$n_2^\alpha$" means the node $n_2$ belongs to the node group in which the nodes are similar to $n_a$. The distance between node $n_2^\alpha$ and node $n_{14}^\ell$ is

$$w(n_2^\alpha, n_{14}^\ell) = d(n_2, n_{14}) + 10d(n_2, n_a) + 10d(n_{14}, n_c)$$
$$= 7 + 10 \times 0.10 + 10 \times 0.22$$
$$= 10.20$$
Table 3 The distance among nodes in the member graph shown in Figure 9. Only potential candidate nodes of input nodes are listed here because only the path lengths between these candidates are considered in the inexact subgraph extraction algorithm.

<table>
<thead>
<tr>
<th></th>
<th>n_2</th>
<th>n_3</th>
<th>n_4</th>
<th>n_6</th>
<th>n_11</th>
<th>n_16</th>
<th>n_14</th>
<th>n_15</th>
<th>n_17</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>n_3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>n_4</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_6</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_11</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_16</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_14</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_15</td>
<td></td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4 Distance between input nodes n_a, n_b, n_c and nodes of the member graph shown in Figure 9. For instance, the similar nodes in the member graph of the input node n_a are (n_2, n_3, n_4). The distance between n_a and node n_2 is \( d = 0.10 \).

<table>
<thead>
<tr>
<th></th>
<th>n_2</th>
<th>n_3</th>
<th>n_4</th>
<th>n_6</th>
<th>n_11</th>
<th>n_16</th>
<th>n_14</th>
<th>n_15</th>
<th>n_17</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_a</td>
<td>0.10</td>
<td>0.05</td>
<td>0.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_b</td>
<td>0.21</td>
<td>0.05</td>
<td>0.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_c</td>
<td>0.22</td>
<td>0.10</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5 The weight of each edge is given by the distance between the two nodes plus 10 times the query node distance. In the table \( \varepsilon = 2.60/3 = 0.86 \). A minimal spanning tree algorithm is applied to the graph based on the edge weights. Notice that there are four candidate nodes (n_3, n_11, n_16, n_17) are selected for three query nodes (n_a, n_b, n_c) because we also consider the compactness of the final system graph.

<table>
<thead>
<tr>
<th></th>
<th>n_2^a</th>
<th>n_3^a</th>
<th>n_4^a</th>
<th>n_6^a</th>
<th>n_11^a</th>
<th>n_16^a</th>
<th>n_14^a</th>
<th>n_15^a</th>
<th>n_17^a</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_2</td>
<td>0</td>
<td>(\varepsilon)</td>
<td>(\varepsilon)</td>
<td>10.20</td>
<td>10.00</td>
<td>9.40</td>
<td>7.10</td>
<td>6.50</td>
<td>11.20</td>
</tr>
<tr>
<td>n_3</td>
<td>0</td>
<td>(\varepsilon)</td>
<td>8.70</td>
<td>8.50</td>
<td>7.90</td>
<td>5.60</td>
<td>9.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_4</td>
<td></td>
<td>(\varepsilon)</td>
<td>9.20</td>
<td>9.00</td>
<td>8.40</td>
<td>6.10</td>
<td>5.50</td>
<td>10.20</td>
<td></td>
</tr>
<tr>
<td>n_6</td>
<td></td>
<td></td>
<td>(\varepsilon)</td>
<td>(\varepsilon)</td>
<td>11.30</td>
<td>4.70</td>
<td>5.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_11</td>
<td></td>
<td></td>
<td></td>
<td>(\varepsilon)</td>
<td>(\varepsilon)</td>
<td>11.10</td>
<td>4.50</td>
<td>5.40</td>
<td></td>
</tr>
<tr>
<td>n_16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\varepsilon)</td>
<td>(\varepsilon)</td>
<td>(\varepsilon)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>n_2^b</th>
<th>n_3^b</th>
<th>n_4^b</th>
<th>n_6^b</th>
<th>n_11^b</th>
<th>n_16^b</th>
<th>n_14^b</th>
<th>n_15^b</th>
<th>n_17^b</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n_16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Given the new distance matrix shown in Table 5, we apply Prim algorithm to get the minimal spanning tree and thus, obtain the optimal candidate nodes in \( G_M \). First, \( w(n_{16}^b, n_{17}^c) = 2.60 \) is the minimum. Thus, we have two candidates node: \( n_{16} \) and \( n_{17} \). The next minimum is \( w(n_{11}^b, n_{17}^c) = 3.90 \). However, since we already have found better candidate nodes for \( n_b \) and \( n_c \), the edge \( (n_{11}^b, n_{17}^c) \) is skipped. Similarly, edge \( (n_{11}^b, n_{15}^c) \) and edge \( (n_{11}^b, n_{14}^c) \) are skipped. Finally, \( (n_{11}^b, n_{5}^c) \) is picked because this edge connects to the third input node \( n_a \).

![Diagram of the system graph](image)

**Figure 10** A system graph is returned from the member graph shown in Figure 3 by querying node \( n_a \), \( n_b \) and \( n_c \). The input nodes have hexagon shapes, and the candidate nodes have circle shapes. The dotted straight lines show the relationships between the input nodes and their corresponding candidate nodes that are used as the input nodes for exact subgraph extraction. The subgraph in the dotted curve line is the result of the system graph of the querying. Notice that query node \( n_b \) has two candidate nodes \( (n_{11}, n_{16}) \) in the member graph that are used as input nodes.

Finally, we have the candidate nodes from \( G_M \) similar to the input node. \( n_3 \) is picked for \( n_a \), \( n_{11} \) and \( n_{16} \) are picked for \( n_b \), \( n_{17} \) is picked for \( n_c \). There are two candidate nodes \( (n_{11}, n_{16}) \) for one input node \( n_b \) because we consider the compactness of the result besides to the similarity. For this reason, sometimes the candidate node may not be the most similar node of the input node. Based on the input
node $S = \{n_5, n_{11}, n_{16}, n_{17}\}$, we can get the system graph as shown in Figure 10 by applying the exact subgraph extraction algorithm to $G_M$ as shown in Figure 3.

### 3.6.3 Data preparation

To measure the similarity or to compute the distance function between two nodes, we needed to collect data outside the graph (i.e. the biological network in MetNetDB). Similarity measurement for genes and proteins are based on gene ontology and similarity of expression patterns. Since we have much less experimental data for metabolites, we currently use the chemical ontology terms of ChEBI (de Matos et al., 2006) to compute the distance function of each metabolite pair.

**Similarity measurements for genes**

Microarray data from the model plant, Arabidopsis, was obtained from NASC arrays (Craigon et al., 2004) and PlexDB (Shen et al., 2005; Tang et al., 2005), and normalized and processed (Mentzen, 2006). Using this data, we calculated the correlations between each pair of genes based on the expression data as revealed by microarray chips (Mentzen, 2006). The top 100 genes with highest correlation values were collected. Next the gene ontology (GO (Harris et al., 2004)) annotation for these genes was used to compute a distance function of each gene pair. GO classifies genes in a directed acyclic graph structure (also called a rooted tree, shown in Figure 11). The first step can filter out most of the 22K genes on the chip. The second step can get an estimate of biologically meaningful similarity values based on the curated GO terms that describe each gene.

There are many similarity measurement methods for GO terms. We obtained the idea of using ontology terms from Bioconductor (Gentleman et al., 2004) GoStats package which is based on the position of the ontology term in the ontology graph; this method is very fast and can be used for any ontology, including both GO and chemical ontology. Speed is critical for the performance of our implementation of inexact subgraph isomorphism and subgraph extraction. The GoStats package
introduces two GO derived distances based on graph similarity: simUI and simLP, which are derived based on a traversal of the ontology graph.

Suppose one gene has multiple terms in GO (Figure 11). These terms are used as the starting nodes for the traversal of the ontology graph (Figure 12). The traversal visits parent nodes until the root node of the ontology is visited. The complete traversal graph is unique for any gene in the gene ontology. Suppose the traversal graphs of node \( x \) and \( y \) are \( G_x = (V_x, E_x) \) and \( G_y = (V_y, E_y) \), let \( G_U = G_x \cup G_y \) and \( G_I = G_x \cap G_y \), simUI is the ratio between the number of nodes in \( G_I \) and the number of nodes in \( G_U \); simLP is the length of the longest path from a leaf node to the root node in \( G_I \) (Figure 12). The two indices (simUI and simLP) measure the similarity between two genes based on the ontology. It is possible that one of them is very small while the other is large. We consider both variables when evaluating the similarity between two genes. In MetNetDB, we use \( \text{simUI} \times \text{simLP} \) as the integrated index.

**Similarity measurement for proteins**

Proteins in MetNetDB are of two types: polypeptide and protein complex. A polypeptide is encoded directly by a gene, thus each polypeptide can be directly mapped to a gene ontology term. A protein complex is composed of one or more polypeptides and in some cases other types of biomolecules. Thus, there are one or more genes encoding a protein complex. Suppose that the genes are \( g_1, g_2, \ldots, g_k \), their traversal graphs in the gene ontology are \( G_1, G_2, \ldots, G_k \), then the traversal graph of the protein complex is \( G_p = G_1 \cup G_2 \cup \cdots \cup G_k \). Thus, for any pair of proteins, we can compute simUI and simLP based on the protein’s traversal graph in the gene ontology.

**Similarity measurement for metabolites**

The chemical ontology ChEBI (de Matos et al., 2006) is not a DAG structure because the curator adds some symmetric relationship types between terms. For instance, the relationship “is conjugate base
of” and “is conjugate acid of” creates a loop between the acid form of one metabolite and the base form of the same metabolite. The method to generate simUI works for the case since it relies only on the node number. To compute simLP, the path should be a simple path. We use shortest path algorithm to get the simple path and the length of the path.

Figure 11 Gene ontology (GO) is a directed acyclic graph (DAG). Each gene has one or more GO terms corresponding to leaf nodes in this DAG. The traversal starts from the leaf nodes to their parent nodes until the root node is reached. The subgraph induced by those visited nodes is the traversal graph of the corresponding gene. In this example, the subgraph in the dotted line is the traversal graph of the gene.

Figure 12 An example of computing simUI and simLP. The traversal graph $G_1$ of “gene 1” contains nodes ($n_8, n_4, n_2, n_1, n_{14}, n_7, n_3$). The traversal graph $G_2$ of “gene 2” contains nodes ($n_{10}, n_6, n_2, n_1, n_{13}, n_7, n_3$). The union of $G_1$ and $G_2$, $G_U$, contains 10 nodes. The intersection of $G_1$ and $G_2$, $G_I$, contains 4 nodes, which is the part in the dotted line. Thus simUI = 4/10 = 0.4. The longest path in $G_I$ has a length 2, thus we have simLP = 2.
3.7 Results

Figure 13 displays the implementation of subgraph extraction in the MetNetDB curator tool. In Figure 13a, biologists first choose highly connected entities. Highly connected entities are biomolecules that participate in many interactions in the database. In the current version of the Arabidopsis biological network database, these are predominately metabolites. As biological knowledge of Arabidopsis expands, regulatory proteins will be included in this category. Before a subgraph is extracted, a subset of these entities should be removed to prevent connections among many pathways. These highly connected entities will be added back to the extracted subgraph. Figure 13b shows how entitywithlocations that the biologists wish to search are specified for a graph query. These entitywithlocations are used to determine the subgraph such that every selected entitywithlocation and any paths between each pair of them are included. Figure 13c shows the pathway selector window. Biologists can use the window to determine the search range. If biologists want to search the whole database, all pathways in the window should be selected.

Figure 14 shows the subgraph extraction results according to the conditions specified in Figure 13a, Figure 13b, and Figure 13c. The red-colored entitywithlocations are the selected entitywithlocations specified by biologists. In this subgraph, all paths between the pair of any two entitywithlocations are included.
Figure 13 An example of query condition for an exact subgraph extraction. a) This is a window for choosing highly connected entities. Users can specify the result number and degree threshold to get the highly connected entities. In current stage, MetNetDB search metabolites only. b) This is a window for choosing entity with locations. These nodes are used as input nodes in subgraph extraction. c) This is a window for choosing pathways. Users can select several pathways as a whole member graph to be searched.
Figure 14 This is the result of the subgraph extraction for the query conditions specified in Figure 13.

Figure 15 shows an example of an inexact subgraph extraction. When the selected entitywithlocations are not in the selected pathways to be searched, the curator tool will find approximate matches for the entitywithlocations based on the similarity scores computed from similarity of RNA accumulation pattern, GO and ChEBI. The correlation data is obtained from Pearson correlations among the 22K genes in pre-processed microarray data (Mentzen, 2006). When the candidate entitywithlocations are determined, the minimal spanning tree algorithm is applied to get a subgraph with respect to both the similarity scores and the compactness of the subgraph. In Figure 15b, the red colored entitywithlocations are those nodes that were specified in the query. The
black colored entitywithlocations are candidates from the MetNetDB graph when the selected entitywithlocations are not available in the selected pathways to be searched.

Figure 15 An example of inexact subgraph extraction. a) The window displays the conditions used to perform the inexact subgraph extraction, including the highly connected metabolites, input entitywithlocations and pathways. b) This window displays the result of the inexact subgraph extraction for the query conditions specified in a). The red colored entitywithlocations are those specified in the query. The black colored entitywithlocations are candidates when the selected entitywithlocations are not available in the selected pathways to be searched.

3.8 Conclusion and future work

The subgraph extraction algorithm introduced in this paper can extract a subgraph defined by a selection of nodes. We have applied this algorithm, combined with Messmer’s subgraph isomorphism algorithm (Messmer and Bunke, 1993; Messmer and Bunke, 1998), to the data integration problem
for MetNetDB, a biological network database. The algorithm is flexible and can be modified for different purposes. For instance, if the path length is very large and needs not to be included in the system graph, the algorithm can be modified by combining it with a \( p \)-neighborhood method.

Subgraph extraction provides a way to get biologically “interesting” sub-networks from a biological network database based on a list of user-selected genes, proteins and metabolites. We proposed to integrate this implementation with our MetNet tools for further network analysis. In the future, this algorithm can be developed for the study of motif evolution in protein-protein interaction networks among various species. In this study, motifs or functional modules of one species will be used to query protein-protein interaction networks of other species. Once the conserved motifs or functional modules are identified, the corresponding proteins or relationships among proteins will be analyzed.

Because structure similarity of metabolites is a good hint for functional similarity, the computation of the similarity score among metabolites can be improved if the structure is included. Our next step is to investigate efficient algorithms for inferring structure similarity between metabolites. The selected algorithm will be integrated into our graph matching applications.

**Acknowledgments**

We are very grateful to Dr. Wieslawa Mentzen for the gene correlation data that are used in the inexact pathway query. We also thank all members of the MetNet group for their valuable input and suggestions. This project is funded by National Science Foundation Arabidopsis 2010 #0520267.

**References**


de Matos P, Ennis M, Darsow M, Guedj M, Degtyarenko K, Apweiler R (2006) ChEBI - Chemical Entities of Biological Interest. NAR Molecular Biology Database Collection entry number 646


CHAPTER 4.  METNETDB, A NETWORK CURATION TOOL FOR BIOLOGISTS

A paper to be submitted to Bioinformatics

Jie Li, Heather L. Babka, Leslie Miller, Nick Ransom, Tian Xia and Eve Syrkin Wurtele

Abstract

Summary: MetNetDB curator tool is a Java-based application for curation of metabolic and regulatory networks. It supports rule checking during pathway edits, has a powerful Boolean query language for pathways and a visual pathway query interface. It also tracks data changes and allows these changes to be reviewed. These features are combined in a labeled graph model to create a database for biological networks with unique curation capabilities. The biological network data can be exported to a platform of MetNet tools.

Availability: http://www.metnetdb.org/MetNet_db.htm

Contact: mash@iastate.edu

4.1 Introduction

In the post-genomic era, biological network databases are critical for biologists to study and model gene functions efficiently using a systems approach. MetNetDB (Wurtele et al., 2007) is a central repository of biological networks for Arabidopsis and other plant species. The curator tool is created to integrate networks from biologists and the literature into MetNetDB.

Numerous pathway visualization and manipulation tools have been developed over the past dozen years. Tools like BioCyc (Karp et al., 2002), PathCase (Ozsoyoglu et al., 2006) and PATIKA (Dogrusoz et al., 2006) provide many useful features for pathway creation, annotation and querying. However, some features important for curation but inherently complicated are not supported in existing tools yet. Examples are: 1) active rule checking during curation; 2) tracking network change
histories; and 3) a visual pathway query system. Here we describe an innovative curator tool, MetNetDB curator tool, which offers these three features and improves curation performance (Fig. 1).

4.2 Novel features of the MetNetDB curator tool

MetNetDB supports browsing, searching, creating and editing pathway structure and annotation. In addition to these essential features, the curator tool provides new features to help biologists edit pathways efficiently.

4.2.1 Active rule checking

Rules are a set of constraints that guarantee data validity in a biological context. For instance, “an enzyme is not a metabolite”. In MetNetDB, rules are classified as biological rules and graph rules. “Catalysis” and “allosteric regulation” are two examples of biological rule topics. Graph rules guarantee the basic data integrity in the context of the graph. For instance, an orphan node or a dangling edge is not permitted. Rules are checked when a curator submits data to the database. The rules are classified as mandatory or optional. Any violation of mandatory rules will block the data submission. In contrast, conflicts with the optional rules only provide a warning to the curator, thus, the curator tool can warn biologists about possible errors, while it still allows exceptional biological data to be saved and curated.

4.2.2 Tracking the history of network changes

Preserving data change history is critical to evaluate the validity of the data. In contrast to conventional changes in annotation of biological entities and interactions, changes in biological networks encompass changes in the graph structure. These changes must be handled separately. MetNetDB curator tool tracks every change in the annotation and the graph structure, including changes in substrates, products, coefficients of interactions, references, interaction lists of pathways,
and molecular locations. The curator tool provides a window to browse the history of data changes in MetNetDB. Biologists can choose one of changes to review in detail or choose one deleted record to undelete.

4.2.3 **Boolean pathway query system and visual pathway query system**

The MetNetDB curator tool supports both Boolean pathway queries and a visual pathway query interface. We implement the Boolean query language for biological entities, interactions and pathways using AND, OR, LIKE, NOT operators, and parentheses. Biologists can query interactions and pathways by inputting biological entities that participate in the interactions and pathways.

In addition to conventional query interfaces, there is a graphical interface for pathway queries. Biologists can draw a partial structure and use the structure to query the similar pathways in the database. This feature allows biologists to determine whether a part of the proposed pathway is already in the database.

4.3 **Implementation**

MetNetDB organizes biological network data as a labeled graph model. In this model, biological entities and interactions are nodes. The relations between entities and interactions are represented as edges. The biological properties of entities and interactions are stored in the node labels, while the edge labels contain the coefficients of the biological entities. The MetNetDB rule checking, data history tracking and graph matching are all built on this labeled graph model.

Active rule checking is performed on the graph structure and labels each time there is an edit operation to the data. These rules are organized in a chain and they can be added or removed easily. Data changing is stored in separate tables with user name and time information. The information can be used to restore the corresponding version if necessary. The pathway query feature is implemented based on Messmer’s inexact subgraph isomorphism algorithm (Messmer and Bunke, 1998). The
matching results are used to extract the common substructures between the input graph and pathways in the database.

The MetNetDB curator tool is open source software. It contains two components, a server application running in the JBoss application server (Johnson, 2005) to interact with MetNetDB and a client application to interact with biologists. The server application manipulates the data and provides labeled graph model based services to all MetNet tools through HiveMind (Johnson, 2005). The client application displays pathways and submits curators edit requests to the server application. This three-tier architecture (user interface, functional logic, and data management) provides a lot of flexibilities for feature expansion with minimal effort.
Fig. 1 Screenshots of MetNetDB curator tool. a) The curator tool detects an error. The network edit window displays a curator’s insertion of NADH as a catalyst of an enzymatic reaction. The inserted NADH conflicts with a rule and is highlighted by a red box (see the upper red ellipse). Because NADH is a metabolite, adding it here conflicts with the rule that an enzyme should be either a polypeptide or a protein complex (see the lower red ellipse). b) The history browser window displays previous edit operations on curator-selected biological network data. If the operation is “delete”, then the “undelete & edit” button will be enabled. Otherwise, if the operation is “insert” or “modify” (see the upper red arrow), the “review & edit” button will be enabled (see the lower red arrow). c) This input window allows biologists to draw a structure and query it in the database. In this example, the curator tries to find a cycle among three metabolites (L-methionine, S-Adomet, and 5-methylthioribose) in the database. d) This window displays a matching result from the structural query in (c). The nodes of the input graph are designated by a red box. The dotted lines indicate the alignment between these nodes of the input graph and the pathway “ethylene biosynthesis and methionine cycle”. For instance, a dotted line indicates that S-Adomet node of the input graph is aligned to the S-Adomet in cytosol (light yellow background) of the pathway. In this pathway we can find a highlighted cycle that inexact matches the input graph.

Acknowledgements

MetNet is supported by NSF Arabidopsis 2010 #0520267.

Conflict of Interest: none declared.
References


CHAPTER 5. GENERAL CONCLUSION

5.1 Summary

Currently MetNetDB provides Arabidopsis networks and annotations for MetNet tools, and integrates regular AraCyc update, and networks from biologists and other databases. In addition, the annotation of biomolecules from a variety of sources is associated with the existing networks in MetNetDB. MetNetDB data can be obtained via HiveMind service (Johnson, 2005) or via web service at http://metnet3.vrac.iastate.edu/api.

The implementation of MetNetDB has demonstrated general steps of data integration for biological network databases. These steps are:

1. Transform external data models of biological networks into the MetNetDB labeled graph model;
2. Define the semantic mapping between the biological objects (biomolecules, interactions and pathways) in the labeled graph model and the importing biological objects;
3. Transform the external biological data into the corresponding biological objects or biological properties in the MetNetDB labeled graph model;
4. Resolve conflicts of data from different sources or different version from a single source.

MetNetDB labeled graph model is designed to organize and integrate biological data into the database. As a graph model, it allows the application of graph analysis. Labels store various types of biological data. Rules help maintain the biological data integrity. Operations define permitted graph transformations. With the implementation of graph matching algorithms aimed for biological network data warehouse including MetNetDB, the external biological processes or pathways can be matched with existing pathways in the local network data warehouse exactly or approximately. Then, the curator can determine how the external network data is imported.
In MetNetDB, the graph matching we have implemented contains two parts, Messmer’s error-correcting subgraph isomorphism algorithm and our algorithm developed for subgraph extraction. The subgraph isomorphism algorithm supports querying an input graph against a graph database. Based on the node mapping found by the isomorphism, our subgraph extraction algorithm extracts the “biologically relevant” subgraphs from the database. Our algorithm can also be used to query a subgraph based on user-specified biomolecules, which may be determined from experiments. The similarity among biomolecules used in the graph matching algorithms is inferred from expression patterns, gene ontology, chemical ontology, and protein-gene relationships. They can be replaced by more advanced and faster algorithms in future.

MetNetDB curator tool is developed to curate MetNetDB efficiently. In addition to the graph visualization and edit features common to most of pathway tools, the curator tool implements several innovative features based on MetNetDB labeled graph model, including the active biological rule checking, tracking data change history and a visual pathway query system. The active biological rule checking system prevents most common errors in curation. Tracking data change history is essential for biological data provenance. The visual pathway query system allows not only computer scientists but also biologists to perform complex graph queries.

With the integration approach, the labeled graph model, the graph matching algorithms and the curator tool developed for MetNetDB, the database can continuously integrate biological data in future.

5.2 Recommendation for future research

MetNetDB is initially designed as an extension of AraCyc network database. Based on the labeled graph model, MetNetDB can be easily expanded to a biological network database for other plants. MetNetDB defines its own controlled vocabularies for biological entity type, interaction type, subcellular location, confidence level, and pathway class. The external terms are transformed to
MetNetDB vocabularies by curators. To ease the transformation process systematically, it would be helpful to setup MetNetDB ontology or adapt existing ontology directly.

The graph matching algorithms implemented in MetNetDB can be applied to many types of biological networks. There are several possible applications of the algorithms. First, the algorithms can be applied to the comparative studies of biological networks such as protein-protein interaction networks and metabolic networks. The comparison of networks of different species can elucidate which module of the biological networks is conserved in the evolution. Second, it is possible to compare the networks from the literature and the networks inferred from experimental data. The comparison can be used to evaluate the validity of an inference algorithm and reveal new structures that are not identified in the past. Third, the graph-matching algorithms provide possibilities to expand network data with text mining results (semi-)automatically in large scale. The text mining technology can extract the hypothetic interaction networks from literature. These hypothetic networks, with appropriate evidence codes assigned, can be integrated into the existing curated networks. The combined networks can either help biologists to evaluate the experimental data more thoroughly, or to ease the curation process.

A problem of the graph matching algorithms (both the subgraph isomorphism and subgraph extraction) is the parameter selection. The parameters for the subgraph isomorphism include the cost function of node insertion and deletion, edge insertion and deletion, and label substitution. The exact subgraph extraction does not need parameters. The parameters for the inexact subgraph extraction include the weight function when computing the overall distance from the distance function of the shortest path and node similarity. All these parameters can affect optimal results. Parameter selection can be conducted by learning process. For example, we can specify various combinations of parameters and show the graph matching results to experts. Once the experts evaluate and adjust the matching results, the optimal parameters can be inferred. Neuhaus (Neuhaus and Bunke, 2007) presents an Expectation Maximization algorithm (Dempster et al., 1977) to derive desired cost
functions for the graph edit based graph-matching algorithms. It still needs to investigate if the algorithm can be applied to MetNetDB given the limited biological networks and large quantity of biomolecules involved in the networks.

References


ACKNOWLEDGMENTS

I owe gratitude to many people for their help to my research.

I am deeply indebted to my major professor, Dr. Eve Wurtele, for her constant support and guidance for my research through the years. I greatly appreciate her leading me into the research area of bioinformatics. I am also indebted to my co-major professor, Dr. Leslie Miller, for his help and advice in the research during the whole PhD study, especially in databases and algorithms. Many thanks to my committee members Dr. Julie Dickerson, Dr. Basil Nikolau, Dr. Shashi Gadia and former committee member Dr. Di Cook. Their advice, time and patience are highly appreciated.

I would like to thank my friends and colleagues, Pan Du, Ling Li, Tian Xia, Lifeng Zhang, Mohammed Alabsi, Wieslawa Mentzen, Heather Babka, Jianling Peng, Yves Sucaet, Nick Ransom and many other people for the discussions and help during the years. In addition, I want to thank all the members in MetNetDB research group for the research environment.

My graduate research has been made possible through the support of National Science Foundation. In addition, I would like to thank Virtual Reality Applications Center for the convenient work environment.

Finally, I owe my best gratitude to my family, Zhenfan Li, Meifang Yuan, Yunxian Zhou, Zheng Li, Jun Li and other family members for their love and constant support.