Chapter 23

Computational Representation of Biological Systems

Zach Frazier, Jason McDermott, Michal Guerquin, and Ram Samudrala

Abstract

Integration of large and diverse biological data sets is a daunting problem facing systems biology researchers. Exploring the complex issues of data validation, integration, and representation, we present a systematic approach for the management and analysis of large biological data sets based on data warehouses. Our system has been implemented in the Bioverse, a framework combining diverse protein information from a variety of knowledge areas such as molecular interactions, pathway localization, protein structure, and protein function.

Key words: Bioverse, data integration, molecular interactions, protein structure, protein function, data warehouse, database, bioinformatics.

1. Introduction

As high-throughput and other large data sets are generated, the ability of researchers to organize and analyze these data will determine the science that can be accomplished. Successful integration of diverse data sources provides novel insight into biological processes. For example, the combination of data sets has been used to discover novel protein–protein interactions in the galactose utilization pathways of yeast *(1, 2)*. In the Bioverse, the application described here, proteins have been annotated with functional descriptions by combining the existing and predicted interaction networks and the existing functional annotations *(3)*.

Integrating biological resources pose many problems for researchers. Resources are designed and developed with a specific user community in mind and, with this specialization, have developed a particular data focus, storage format, and query interface.

Jason McDermott et al. (eds.), *Computational Systems Biology, vol. 541*

DOI 10.1007/978-1-59745-243-4_23

Developing tools to utilize these resources demands both an investment of time and often specific knowledge of the resource. Objects of interest have different identifiers in different contexts, complicating accurate integration. Independent projects collect different information for similar data sets, and may use different standards of measurement. The query interfaces provided for the resource may be restrictive, not allowing for novel uses. For example, using web sites for blast queries to find similar proteins is reasonable for a handful of interesting proteins, but for a large data set it is easier to perform the queries against a local database.

The focus of many biological databases is necessarily narrow, either focused exclusively on single organisms, such as Wormbase *(4)*, databases of structures *(5, 6)*, or pathways *(7)*. Manually integrating the results from many data sources may be feasible for focused questions or small studies, but is time-consuming for large data sets. Several projects have attempted to solve this problem, acting as an intermediary between databases, thereby solving the problem of integration; however, since these often work through the interfaces provided, the throughput of this approach is limited. Services such as BioMoby *(8)*, REMORA *(9)*, and the Bioinformatics Resource Manager *(10)* successfully integrate a variety of data sources and bioinformatics tools. These are excellent resources for small queries across many different databases.

For larger projects, we instead integrate the entire resource. We begin with the raw data provided by the resource maintainers, and develop our own storage system integrated with other data sources based on data warehousing principles.

Data warehouses are an approach to data integration and management, which is used for a variety of problem domains. In addition to maintaining a highly flexible storage system for data, data warehouses allow for the expression of complex relationships and ease the construction and execution of complex queries.

The solutions developed in the Bioverse *(11)* integrate a wide variety of biological data sources, allowing for exploration and prediction of functional, structural, and sequence-based data analysis.

2. Data Warehouses

Data warehouses organize data for analysis and data mining applications. Although they are built on relational database technology, data warehouses differ from traditional online transaction processing (OLTP) databases. Instead, they are designed to support online analytical processing (OLAP). OLTP systems typically support many concurrent users inserting, deleting, and modifying small amounts of data. OLAP systems provide management and

processing of multidimensional data for analysis. The structure and organization of the data models are different for each application type. Data warehouses are built on an OLAP model. An excellent review of data warehouses is Kimball *(12)*.

2.1. Relational Databases

At the center of the data management system is the relational database. Relational algebra was introduced by Codd *(13)*. A large software industry based on his work quickly appeared, and a query language based on relational algebra, Structured Query Language(SQL), has become the standard for most commercial relational databases.

For our purposes, a database is a collection of tables, indexes, relations, and functions. The tables are collections of objects with identical attributes. The attributes are represented as the columns of the tables, and the objects are stored as the rows of the table. Data indexing and custom database functions optimize the access patterns. Interactions with the system are based on transactions, which guarantee the data integrity in the face of unexpected system or process failures. Transactions represent a fundamental atomic action in the database. In the event of an error a transaction is aborted and all changes can be rolled back to assure data consistency.

2.2. Dimensional Models

As specializations of relational databases, the distinguishing feature of a data warehouses is the organization of the data. Traditional OLAP database design methodology focuses on normalized tables. Normalization provides logical separation of data, moving all redundant data to tables, which are referenced as foreign keys. Since the activity in these databases consists of many small transactions, normalization localizes the effects of changes on the database. This design goal is relaxed for data warehouses that have different requirements. Normalization is sacrificed for expressive and efficient queries across large data sets.

Query writing for the dimensional model is straightforward. Queries can easily be constructed, since the relationships between tables are simple and designed for flexibility. Queries against the central fact table will use filters on the linked dimension tables to narrow the focus of the query.

Using a data warehouse provides several benefits. The approach makes the information accessible to more general queries than traditional data schemas, and it is flexible with changes and updates to the underlying data model having minimal impact on the data model organization. New data types can be added without disturbing the existing table structure, and new dimensions can be added to a fact with minimal disruption.

2.2.1. Facts

Facts are the data points of the system. They are the generated or computed measurements that are the focus of the representation, and are defined in terms of the measurement conditions and

parameters. Each fact type is stored in a corresponding "Fact Table". Fact tables are large, many containing millions of rows. Several columns of the fact table will be foreign keys, describing in detail "dimensions" of the facts. Other columns will be numeric or categorical data, the "measures" of the fact. The redundancy of storing most of the data in a large central table, with a handful of satellite tables, allows for flexibility at the cost of some redundancy. As an example of a fact table, consider the molecule_sequence table of **Fig. 23.2**.

2.2.2. Dimensions

Dimensions represent the complex attributes of the facts. These columns of the fact table are pointers to other tables or foreign keys. These are the features of facts that themselves have many features, which would be useful query filters. The features are stored in "dimension tables" that describe a particular feature of the fact in detail. These tables rarely change, and encapsulate a small set of specific data.

2.2.3. Measures

Measures are the parameters that make up the facts. These are discrete values and are usually numeric and additive. Being additive allows for summary queries on the fact table. Simpler then dimensions, these do not have associated attributes. These are simple columns of the fact table.

2.2.4. Star Schemas

This organization yields a "star schema" with the fact table at the center, surrounded by many dimension tables. These structures are the goal of data warehouse design. While not highly normalized like many database designs, the star schema allows for complex queries to be made over fact tables efficiently. Filters on the associated dimension tables and measures provide a flexible constraint-based search system, which can adapt to a wide variety of questions, allowing researchers to identify and isolate relevant facts of interest. In some cases the star schema may have a depth of more than a single table. These snowflake schemas, although sometimes necessary, should be avoided, as they make query writing complicated and can impact performance.

A data warehouse will contain several fact tables, which may share dimensions. Each fact table and the associated dimensions are considered a distinct "data mart". Generally, the data warehouse will consist of several independent data marts, which have an independent focus, but which share a few dimension tables.

There are many efforts to standardize data representation in systems biology. Systems Biology Markup Language (SBML) *(14)* is a data exchange format designed for pathway

and network models. Another particularly successful example is the PSI-MI *(15)* format for interaction data sets. In the cases where a well-defined data specification exists, it is easier to design the database tables with the given specification as a guide. While this method may not provide a true warehouse design, it is a step toward the goal of integration and efficiency. The staging data of the Bioverse, for example, closely mirrors in structure the PSI-MI format.

3. Data Warehouse Construction

Development of the data warehouse from source data to completed database is an integrated process which includes both the development of a data model and the development of the tools required to load large quantities of data.

3.1. Model Design

3.1.1. Facts and Granularity

The first decision to be made in model design is the focus of the fact tables. Collecting and storing data at the wrong resolution will impact the ability of the warehouse to answer research questions. If highly specific data are collected, it may be impossible to construct queries on relevant aggregates. On the other hand, queries will not be able to filter well if the fact tables or the dimension tables are too general.

3.1.2. Dimensions

The characteristics of the fact table are stored in the dimension tables. The choice of columns in the dimension tables determines the queries that are supported. Verbosity and redundancy are acceptable since support for rich analysis is the goal. Although storage space is a consideration for large databases, the dimension tables even without normalization will not represent a major storage problem. Typically, the fact tables that are relatively compact will have orders of magnitudes more rows than dimension tables.

3.2. Extracting, Transforming, and Loading

The migration of data from its native source format into a warehouse is commonly referred to as the Extraction, Transformation, and Loading step (ETL). Data can be extracted from other databases or text files.

The design and implementation of ETL tools is one of the most time-consuming aspects of data warehouse development. Complex rules and transformations must be applied and errors must be dealt with intelligently. Many of these steps take place in a staging area of the warehouse. **Fig. 23.1**.

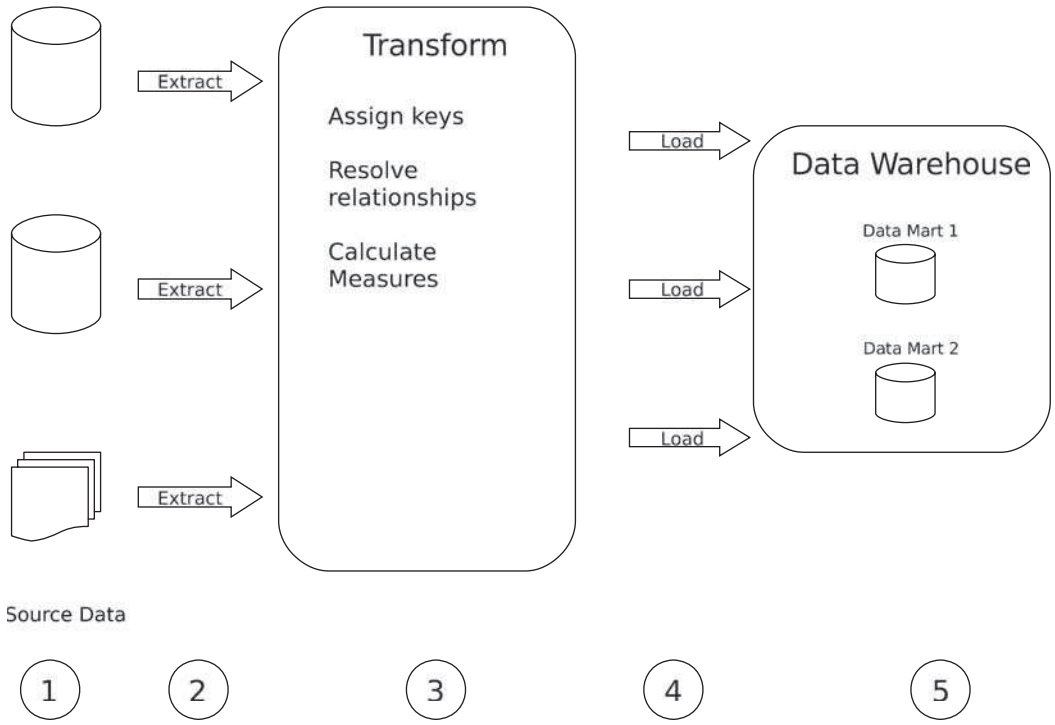


Fig. 23.1. The ETL process. (1) Data sources such as downloaded versions of projects, and the results of bioinformatics tools that have been run against local data sets. (2) All of these data are extracted from their original storage formats and parsed. (3) The data are transformed. Additional fields are calculated, lookups to the database are made, and data sets are validated against existing data. (4) The data are loaded in the data warehouse. (5) The data exist in a set of data marts, which are accessed from a range of applications and analytical tools.

3.2.1. Extraction

Programs process the source data, extracting the relevant information and filtering data that will not be used or have already been loaded. The data will be moved to either text files or to a separate area of the database, depending on the transformation steps necessary.

3.2.2. Transformation

Transformations cover many different facets of data management and are often the most complex part of the ETL process. A variety of operations are carried out in the transformation step. Validation and verification ensure that the data are well formed and adhere to any range or value constraints and that, if attributes are referenced, new attributes of the data can be computed based on the existing data. Data can be grouped and merged or split to provide for data at a more appropriate scale. Lookups can be made against a database to fill in a variety of columns such as foreign key values, or unique ids generated by the database.

3.2.3. Loading

Finally, when all of the data are prepared, they are loaded into the data warehouse. This step is optimized for speed. Lookups in the database are avoided, having been accomplished in the transformation step.

3.2.4. Metadata

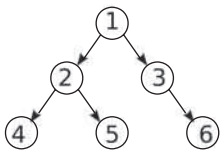
It is important to keep track every piece of data during processing. In the Bioverse we track a number of features of data including its source, what transformations have been carried out, any errors or warnings encountered, time required to process, and more. This metadata provides important information that helps with data management. In our case this is stored in the database where all the ETL tools responsible for data processing can log activity in relation to the data sets they operate on.

4. Bioverse Model Design

The Bioverse includes a variety of information types as well as sources. Interaction data, functional annotations, structure classification, and sequence similarity data types are present. Each type of data requires unique tools for ETL as well as distinct storage in the staging area and the warehouse. Here we will look at a handful of tables and discuss the design choices that were made.

4.1. Sequence Data

Sequence data is stored in the molecule_sequence table (**Fig. 23.2**), which contains information about the type of the sequence as well as dimensions about the source of the sequence.



Transitive closure table

Parent	Child	Distance	Is root	Is leaf
1	2	1	T	F
1	4	2	T	T
1	5	2	T	T
2	4	1	F	T
2	5	1	F	T
1	3	1	T	F
3	6	1	F	T
1	6	2	T	T

Adjacency Table

Parent	Child
1	2
1	3
2	4
2	5
3	6

Fig. 23.2. A portion of the molecule_sequence data mart. The molecule sequence table is the central fact. There are three-dimension tables shown, **`taxon`**, **`molecule_type`**, and **`alphabet`**. Each of these has many interesting features, which can be used to limit searches on the **`molecule_sequence`** table. Additionally, the molecule_sequence table includes several measures, such as **`seq_len`** shown as the last visible column.

A number of indices are also created on the columns of the sequence data and the associated dimensions to accommodate fast queries of features. The MD5 hash of the sequence is stored for fast exact matches, along with a hash of the first 10 and the reverse of the last 10 residues in the case of proteins for fast matching of the start and end of sequences.

4.2. Taxonomy Data: A Hierarchy Example

Hierarchical data occurs throughout the Bioverse. Representation of these structures is particularly difficult in relational databases. The choice of representation scheme depends on the relevant queries. The most straightforward storage solution is to store all parent–child relationships of the hierarchy. This is efficient in storage space, but only allows for a narrow range of queries about table structure. Different table structures are more useful for queries about depth and relationships between entities in the hierarchy. Here we present an approach for storing hierarchical data in the database, which is based on the topological closure of the paths in the hierarchy. This storage mechanism, which is relatively large and expensive to compute, provides a fast mechanism for a wide variety of hierarchical queries. This approach is described in Chapter 5.6 of Kimball *(12)*. Several dimensions in the Bioverse are structured in hierarchies including the Gene Ontology *(16)*, SCOP *(17)*, and the NCBI taxonomy database *(18)*. In each case we use this technique to increase the potential filters on these dimensions.

The taxonomy data provided by the NCBI is used by many different data marts in the Bioverse. As a dimension it needs to be filtered in several ways. A simple list of taxon entries is stored in the taxon table in **Table 23.1**. To provide for more complex queries about relative positions, we calculate the topological closure of the taxonomy tree. This closure is a collection of all paths in the tree. We record the ancestor and child node, as well as the distance between them, and other information about their place in the tree. An example topological closure calculation is given in **Fig. 23.3**. Depending on the query, a fact table can refer to either the original table or the table holding the topological closure. Since the topological closure table has an additional reference to the original table, certain filters will go through two tables, a use of a snowflake schema.

When representing the topological closure, additional columns are necessary whether or not this path is the shortest path, the number of paths between these nodes, and the number of shortest paths between the nodes. While this information is redundant for the taxonomy example where every node only has a single parent, in the event of more complex topologies, these columns are useful for filtering as well. As an example the Gene Ontology hierarchy allows for many parents, making it a directed acyclic graph (DAG) rather than a tree.

Table 23.1
The basic taxon table. No hierarchical information is present. This is a simple dimension table describing taxonomy entries. Columns such as `division_id`, `genetic_code_id`, and `mito_genetic_code_id` refer to other tables built on data provided by the taxonomy database. The various ranks such as kingdom, phylum, and family are stored for each entry to allow for queries about subtree position. To enable some tree queries, relevant information such as `is_leaf` is in the `taxon` table. This table can be used to search for nodes at a known rank depth, or which has a certain ancestor. Relative queries are not available from this table

Taxon table

Column	Description
`Taxon_id`	Primary Key
`parent_taxon_id`	Reference to the parent node.
`rank`	NCBI rank, genus, species, etc.
`division_id`	NCBI division
`genetic_code_id`	Codon table
`mito_genetic_code`	Codon table of mitochondria
`embl_code`	EMBL 2 letter code.
`is_leaf`	If this is a leaf node
`is_root`	If this is the top of the tree
`distance_to_root`	Number of elements to the root
`distance_to_leaf`	. . .
`kingdom`	
`phylum`	
...	. . .
`genus`	
`species`	
`scientific_name`	Common name
`other_names`	A hash of other names

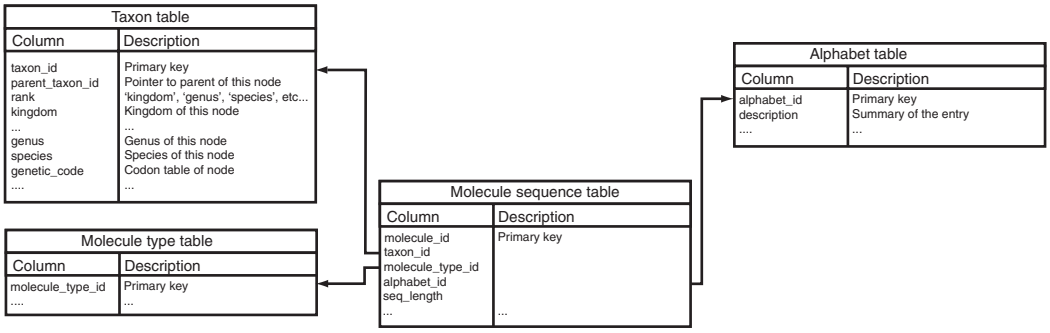


Fig. 23.3. An example of the topological closure data stored for a tree. All paths in the original tree are enumerated in the topological closure.

In addition to hierarchies, graphs and networks are common structures in biological systems including protein–protein interaction networks *(19–22)*, biochemical pathways *(23)*, and others. However, the techniques outlined for trees and directed acyclic graphs are no longer appropriate for graphs.

Answering any more than very basic graph queries is hard in relational databases. One approach is to use a specialized application, which is designed for graph and network representation and queries. In the Bioverse database we have implemented such an application at the interface between the database and the application. It supports a simple breadth first search, as well as searches for graph motifs. While this makes certain tasks simple, specialized applications will not alleviate all problems. Finding the shortest path between two nodes is an expensive operation that may be better done outside of the database. The Bioverse does not have a pressing need for these types of expensive queries, so they have not yet been implemented.

4.3. Functional Data: Affinity Grouping Example

Combining or comparing data from different rows of the same table is expensive in relational databases. This is one reason why the choice of granularity is so important for the fact tables. As previously discussed if the chosen resolution is too fine, queries will require comparing rows. In some cases, there is no alternative because the queries are at many scales. One approach for storing data at different resolutions is affinity grouping, which enables the analysis of individual records as well as certain groups of records.

In the Bioverse, we have both functional annotations from trusted sources and predicted functional annotations.

Table 23.2
A simple record of GO annotations

GO annotation table

Column	Description
`taxon_id`	Foreign Key to the Taxon table
`molecule_id`	The identifier of the molecule
`go_id`	Foreign Key to the Gene Ontology table
`confidence`	Confidence assigned to the annotation

A simple table for the GO annotations stored in the Bioverse is shown in **Table 23.2**. While this table provides information on protein annotations, it is a limited view of the data. Since each protein annotation is a unique fact, it is not easy to see which annotations are related or which are the most specific annotations in the DAG of GO features. To provide access to this data we have calculated a second table that gives information about which combinations of GO annotations are seen together. An added benefit to calculating a priori this data is having access to summary statistics when making queries of the affinity-grouped tables. It is desirable to have information on frequencies, which can be used to estimate the statistical significance of an observation. This affinity-grouping table is shown in **Table 23.3**.

4.3.1. Audit Dimensions

Data that are unrelated to the science of the measurements, but remains relevant, can be annotated in audit dimensions. These tables record events that are significant, interesting, or possible errors. An audit dimension can provide an overview of the fact table or relevant meta data about the measurement. For computed values an audit dimension may be used to record the version information of software or runtime parameters used to generate the fact.

For numeric columns audit dimensions may be used to mark data that are missing, provide meanings or justifications for missing data, or to identify interesting data. A flag that marks all data which is more than two standard deviations away from the mean allows the identification of problematic or interesting data cases.

Table 23.3
The `pair_count` column gives the total number of times these two annotations are seen together, and the `go_1_count` and `go_2_count` give how often annotation occurs independently. This summary table is very useful for exploring annotation pairs and other relationships. The `is_related` column stores whether one annotation of the two is an ancestor of the other in the DAG, in which case the relationship represents the frequency of different parts of the subtree of that node

GO and GO affinity grouping table

Column	Description
`taxon_id`	Foreign Key to the Taxon table
`go_id_1`	Foreign Key to the Gene Ontology table
`go_id_2`	Foreign Key to the Gene Ontology table
`pair_count`	number of co-occurrences of `go_id_1` and `go_id_2` in molecules
`go_1_count`	occurrences of go_id_1 in organism
`go_2_count`	occurrences of go_id_2 in organism
`is_related`	Whether one of the go identifiers is a parent of the other
`distance`	The distance between the entries in the GO tree

5. Bioverse Extraction, Transformation, and Loading

For the Bioverse we have developed in-house data-processing tools, which move the data through a pipeline architecture, processing and running algorithms at each stage.

5.1. ETL Discussion

There are a few guiding principles that influence the design of our tools:

1. Process data in large blocks. Since frequent disk access is expensive, the data are read and processed in blocks that fit into memory.
2. Filter early and often. We filter data as quickly as possible. After opening a block of data, the first steps that are taken are to eliminate any data that are not of interest, in order to minimize later workloads.

3. Expect failures and corrupted, ambiguous, and inconsistent data. All of the operations performed on the data are recorded. Any corrupted, ambiguous, and inconsistent data that are encountered will need to be dealt with robustly.
4. Log and audit every block, and store all bad data in a way that allows for restarting for failed blocks.

The ETL tool when constructed allows for short programs to be written, which will perform on disk translation of the downloaded and generated data. This data should leave in a state that is amenable to being uploaded to the database.

5.2. Bioverse Extraction

Bioverse data is extracted from a wide variety of data formats. For each data format a parser is available, which will produce all data in the file without any attempt at filtering or validation. Parsers catch formatting errors quite often, and log the failure of the data read to the database. Additionally any data that was expected or is optional that was not present is marked as a failure or a missing value.

5.3. Bioverse Transformation

The data generated from the extraction stage are transformed to prepare it for the Bioverse staging area. Blast hits against databases are checked to make sure that the matching molecule is actually in the database we have loaded. Many types of data will have unique values generated so they can be identified unambiguously later. Errors encountered at this stage are recorded in the database, and the data at fault is not processed further. If the offending program is restarted, it will not duplicate data, since everything takes place in a single database transaction and the failure aborts the transaction.

5.4. Bioverse Loading

The loader code works with large chunks of data, each of which is associated with a block of records. These data are copied into the database. In the event of failure, all of the records in the block are marked as failed in the database, and none are loaded. Again, this consistency ensures that the operation can be repeated when the problems with the data have been resolved, or the underlying data have been regenerated, and there will be no duplication of data.

6. Conclusion

We have addressed the complex problem of creating a database and representation to store a wide variety of biological information.

We expect that, in conjunction with the Pipeline, API, and web application, the database model we have created will be useful to help generate hypotheses and solve problems for biological research.

In addition to introducing the basic data warehousing concepts, we have provided a general strategy for the management and integration of biological data. Specific examples demonstrate how the Bioverse is constructed and how large volumes of data are loaded, stored, and analyzed within the data warehouse.

References

1. Hwang, D., Rust, A. G. G., Ramsey, S., Smith, J. J. J., Leslie, D. M. M., Weston, A. D. D., et al. A data integration methodology for systems biology. *Proc Natl Acad Sci U S A,* 2005, 102(48):17296–17301.
2. Hwang, D., Smith, J. J., Leslie, D. M., Weston, A. D., Rust, A. G., Ramsey, S., et al. A data integration methodology for systems biology: Experimental verification. *Proc Natl Acad Sci U S A* 2005,102(48); 17302–17307.
3. McDermott, J., Bumgarner, R., & Samudrala, R. Functional annotation from predicted protein interaction networks. *Bioinformatics* 2005,21(15):3217–3226.
4. Chen, N., Harris, T. W., Antoshechkin, I., Bastiani, C., Bieri, T., Blasiar, D., et al. (2005). WormBase: A comprehensive data resource for caenorhabditis biology and genomics. *Nucleic Acids Res* 2005,33(Supplement 1):D383.
5. Haft, D. H., Selengut, J. D., & White, O. The TIGRFAMs database of protein families. *Nucleic Acids Res* 2003,31(1): 371–373.
6. Madera, M., Vogel, C., Kummerfeld, S. K., Chothia, C., & Gough, J. The SUPERFAMILY database in 2004: Additions and improvements. *Nucleic Acids Res* 2004,32: D235–D239.
7. Kanehisa, M., Goto, S., Kawashima, S., Okuno, Y., & Hattori, M.. The KEGG resource for deciphering the genome. *Nucleic Acids Res* 2004, 32(Database issue)
8. Wilkinson, M. D., & Links, M. BioMOBY: An open source biological web services proposal. *Brief Bioinform* 2002,3(4):331–341.
9. Carrere, S., & Gouzy, J. REMORA: A pilot in the ocean of BioMoby web-services. *Bioinformatics* 2006,22(7): 900–901.
10. Shah A..R, Singhal M., Klicker K. R., Stephan E. G., Wiley H. S., & Waters K. M. Enabling high-throughput data management for systems biology: The Bioinformatics Resource Manager. *Bioinformatics* 2007,23(7):906–909.
11. McDermott, J., Guerquin, M., Frazier, Z., Chang, A., & Samudrala, R. BIOVERSE: Enhancements to the framework for structural, functional, and contextual annotations of proteins and proteome. *Nucleic Acids Res* 2005,33:W324–W325.
12. Kimball, R., Ross, M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling 2002, Wiley, New York, NY.
13. Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM* 1970,13(6):377–387.
14. Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., et al. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics* 2003,19(4): 524–531.
15. Hermjakob, H., Montecchi-Palazzi, L., Bader, G., Wojcik, J., Salwinski, L., Ceol, A., et al. The HUPO PSI's molecular interaction format – a community standard for the representation of protein interaction data. *Nat Biotechnol* 2004,22(2):177–183.
16. Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., et al. Gene ontology: Tool for the unification of biology. The gene ontology consortium. *Nat Genet* 2000,25(1):25–29.
17. Lo Conte, L., Ailey, B., Hubbard, T. J., Brenner, S. E., Murzin, A. G., & Chothia, C. SCOP: A structural classification of proteins database. *Nucleic Acids Res* 2000, 28(1);257–259.
18. Benson D.A., Karsch-Mizrachi I., Lipman D.J., Ostell J., Wheeler D.L. GenBank: Update. *Nucleic Acids Res* 2004,32: D23–D26.

19. Bader, G. D., Betel, D., & Hogue, C. W. BIND: The biomolecular interaction network database. *Nucleic Acids Res* 2003,31 (1):248–250.
20. Breitkreutz, B. J., Stark, C., & Tyers, M. The GRID: The general repository for interaction datasets. *Genome Biol* 2003 4(3):R23.
21. Chatr-aryamontri, A., Ceol, A., Palazzi, L. M., Nardelli, G., Schneider, M. V., Castagnoli, L., et al. MINT: The molecular INTeraction database. *Nucleic Acids Res* 2007,35:D572–D574.
22. Xenarios, I., Rice, D. W., Salwinski, L., Baron, M. K., Marcotte, E. M., & Eisenberg, D. DIP: The database of interacting proteins. *Nucleic Acids Res* 2000,28(1): 289–291.
23. Kanehisa, M., Goto, S., Kawashima, S., Okuno, Y., & Hattori, M. (2004). The KEGG resource for deciphering the genome. *Nucleic Acids Res* 32(Database issue).